INTERNATIONAL STANDARD

ISO/IEC 9945-2

System Interfaces,
IEEE Std 1003.1™, 2003 Edition
The Open Group Technical Standard
Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

Third edition
2003-08-15

Information technology — Portable Operating System Interface (POSIX®) —

Part 2: System Interfaces

Technologies de l'information — Interface pour la portabilité des systèmes (POSIX®) —

Partie 2: Interfaces systèmes

Reference number
ISO/IEC 9945-2:2003(E)
System Interfaces, IEEE Std 1003.1, 2003 Edition
The Open Group Technical Standard, Issue 6

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

ISO/IEC 9945-2:2003(E)
System Interfaces, IEEE Std 1003.1, 2003 Edition
The Open Group Technical Standard, Issue 6

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO

Case postale 56 • CH-1211 Geneva 20

Tel. + 41 22 749 01 11

Fax + 41 22 749 09 47

E-mail copyright@iso.org

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

ISO/IEC 9945-2:2003(E)

IEEE Std 1003.1™, 2003 Edition

The Open Group Technical Standard
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

Information Technology — Portable Operating System Interface (POSIX®)

Part 2: System Interfaces

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

and

The Open Group

Adopted as an International Standard by the International Organization for Standardization and by the International Electrotechnical Commission

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

International Standard ISO/IEC 9945-2:2003(E)

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 9945 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 9945-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This third edition of ISO/IEC 9945-2 is a minor revision and, together with ISO/IEC 9945-1, ISO/IEC 9945-3, and ISO/IEC 9945-4, cancels and replaces ISO/IEC 9945-1:2002 and ISO/IEC 9945-2:2002, ISO/IEC 9945-3:2002 and ISO/IEC 9945-4:2002.

ISO/IEC 9945 consists of the following parts, under the general title *Information technology — Portable Operating System Interface (POSIX®)*:

- *Part 1: Base Definitions*
- *Part 2: System Interfaces*
- *Part 3: Shell and Utilities*
- *Part 4: Rationale*



International Organization for Standardization/International Electrotechnical Commission
Case postale 56 • CH-1211 Genève 20 • Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Abstract

This standard is simultaneously ISO/IEC 9945:2003, IEEE Std 1003.1-2001, and forms the core of the Single UNIX Specification, Version 3.

The IEEE Std 1003.1, 2003 Edition includes IEEE Std 1003.1-2001/Cor 1-2002 incorporated into IEEE Std 1003.1-2001 (base document). The Corrigendum addresses problems discovered since the approval of IEEE Std 1003.1-2001. These changes are mainly due to resolving integration issues raised by the merger of the base documents that were incorporated into IEEE Std 1003.1-2001, which is the single common revision to IEEE Std 1003.1™-1996, IEEE Std 1003.2™-1992, ISO/IEC 9945-1:1996, ISO/IEC 9945-2:1993, and the Base Specifications of The Open Group Single UNIX® Specification, Version 2.

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

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

The following areas are outside the scope of this standard:

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

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

Keywords

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

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Copyright © 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. All rights reserved. This printing is by the International Organization for Standardization with special permission of the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. Published in Switzerland.

System Interfaces, Issue 6

Published 31 March 2003 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, U.S.A.
ISBN: 0-7381-3436-8 PDF 0-7381-3564-X / SS95078 CD-ROM 0-7381-3563-1 / SE95078
Printed in the United States of America by the IEEE.

Published 31 March 2003 by The Open Group
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K.
Document Number: C032
ISBN: 1-931624-24-0
Printed in the U.K. by The Open Group.

All rights reserved. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission from both the IEEE and The Open Group.

Portions of this standard are derived with permission from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

Permissions

Authorization to photocopy portions of this standard for internal or personal use is granted provided that the appropriate fee is paid to the Copyright Clearance Center or the equivalent body outside of the U.S. Permission to make multiple copies for educational purposes in the U.S. requires agreement and a license fee to be paid to the Copyright Clearance Center.

Beyond these provisions, permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests to the copyright holders should be sent by email to *austin-group-permissions@opengroup.org*.

Feedback

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

IEEE

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property, or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "AS IS".

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with the IEEE.[1] Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

A patent holder has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and non-discriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Authorization to photocopy portions of any individual standard for internal or personal use is granted in the U.S. by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to the Copyright Clearance Center.[2] Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center. To arrange for payment of the licensing fee, please contact:

Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923, U.S.A., Tel.: +1 978 750 8400

Amendments, corrigenda, and interpretations for this standard, or information about the IEEE standards development process, may be found at *http://standards.ieee.org*.

Full catalog and ordering information on all IEEE publications is available from the IEEE Online Catalog & Store at *http://shop.ieee.org/store*.

1. For this standard, please send comments via the Austin Group as requested on page iii.
2. Please refer to the special provisions for this standard on page iii concerning permissions from both copyright holders and arrangements to cover photocopying and reproduction across the world, as well as by commercial organizations wishing to license the material for use in product documentation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The Open Group's mission is to offer all organizations concerned with open information infrastructures a forum to share knowledge, integrate open initiatives, and certify approved products and processes in a manner in which they continue to trust our impartiality.

In the global eCommerce world of today, no single economic entity can achieve independence while still ensuring interoperability. The assurance that products will interoperate with each other across differing systems and platforms is essential to the success of eCommerce and business workflow. The Open Group, with its proven testing and certification program, is the international guarantor of interoperability in the new century.

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group's members include some of the largest and most influential organizations in the world. The flexible structure of The Open Groups membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information is available on The Open Group web site at *http://www.opengroup.org*.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes the *Westwood* family of tests for this standard and the associated certification program for Version 3 of the Single UNIX Specification, as well tests for CDE, CORBA, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at *http://www.opengroup.org/testing*.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at *http://www.opengroup.org/pubs*.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at *http://www.opengroup.org/corrigenda*.

Full catalog and ordering information on all Open Group publications is available at *http://www.opengroup.org/pubs*.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Contents

Chapter	**1**	**Introduction**	**1**
	1.1	Scope	1
	1.2	Conformance	1
	1.3	Normative References	1
	1.4	Change History	1
	1.5	Terminology	1
	1.6	Definitions	3
	1.7	Relationship to Other Formal Standards	3
	1.8	Portability	3
	1.8.1	Codes	3
	1.9	Format of Entries	11
Chapter	**2**	**General Information**	**13**
	2.1	Use and Implementation of Functions	13
	2.2	The Compilation Environment	13
	2.2.1	POSIX.1 Symbols	13
	2.2.1.1	The _POSIX_C_SOURCE Feature Test Macro	14
	2.2.1.2	The _XOPEN_SOURCE Feature Test Macro	14
	2.2.2	The Name Space	14
	2.3	Error Numbers	21
	2.3.1	Additional Error Numbers	28
	2.4	Signal Concepts	28
	2.4.1	Signal Generation and Delivery	28
	2.4.2	Realtime Signal Generation and Delivery	29
	2.4.3	Signal Actions	30
	2.4.4	Signal Effects on Other Functions	34
	2.5	Standard I/O Streams	34
	2.5.1	Interaction of File Descriptors and Standard I/O Streams	35
	2.5.2	Stream Orientation and Encoding Rules	36
	2.6	STREAMS	38
	2.6.1	Accessing STREAMS	39
	2.7	XSI Interprocess Communication	39
	2.7.1	IPC General Description	40
	2.8	Realtime	41
	2.8.1	Realtime Signals	41
	2.8.2	Asynchronous I/O	41
	2.8.3	Memory Management	43
	2.8.3.1	Memory Locking	43
	2.8.3.2	Memory Mapped Files	43
	2.8.3.3	Memory Protection	43
	2.8.3.4	Typed Memory Objects	44
	2.8.4	Process Scheduling	44

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.8.5 Clocks and Timers 48
2.9 Threads 50
2.9.1 Thread-Safety 50
2.9.2 Thread IDs 51
2.9.3 Thread Mutexes 51
2.9.4 Thread Scheduling 52
2.9.5 Thread Cancellation 54
2.9.5.1 Cancelability States 54
2.9.5.2 Cancellation Points 55
2.9.5.3 Thread Cancellation Cleanup Handlers 57
2.9.5.4 Async-Cancel Safety 57
2.9.6 Thread Read-Write Locks 58
2.9.7 Thread Interactions with Regular File Operations 58
2.10 Sockets 58
2.10.1 Address Families 58
2.10.2 Addressing 59
2.10.3 Protocols 59
2.10.4 Routing 59
2.10.5 Interfaces 59
2.10.6 Socket Types 59
2.10.7 Socket I/O Mode 60
2.10.8 Socket Owner 60
2.10.9 Socket Queue Limits 60
2.10.10 Pending Error 60
2.10.11 Socket Receive Queue 61
2.10.12 Socket Out-of-Band Data State 61
2.10.13 Connection Indication Queue 62
2.10.14 Signals 62
2.10.15 Asynchronous Errors 62
2.10.16 Use of Options 63
2.10.17 Use of Sockets for Local UNIX Connections 66
2.10.17.1 Headers 66
2.10.18 Use of Sockets over Internet Protocols 66
2.10.19 Use of Sockets over Internet Protocols Based on IPv4 67
2.10.19.1 Headers 67
2.10.20 Use of Sockets over Internet Protocols Based on IPv6 67
2.10.20.1 Addressing 67
2.10.20.2 Compatibility with IPv4 68
2.10.20.3 Interface Identification 68
2.10.20.4 Options 69
2.10.20.5 Headers 70
2.11 Tracing 70
2.11.1 Tracing Data Definitions 71
2.11.1.1 Structures 71
2.11.1.2 Trace Stream Attributes 75
2.11.2 Trace Event Type Definitions 76
2.11.2.1 System Trace Event Type Definitions 76
2.11.2.2 User Trace Event Type Definitions 79

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.11.3 Trace Functions 79
2.12 Data Types 80

Chapter 3 System Interfaces 83

Index 1669

List of Tables

2-1 Value of Level for Socket Options 63
2-2 Socket-Level Options 64
2-3 Trace Option: System Trace Events 77
2-4 Trace and Trace Event Filter Options: System Trace Events 78
2-5 Trace and Trace Log Options: System Trace Events 78
2-6 Trace, Trace Log, and Trace Event Filter Options: System Trace Events 79
2-7 Trace Option: User Trace Event 79

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

Structure of the Standard

This standard was originally developed by the Austin Group, a joint working group of members of the IEEE, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1, as one of the four volumes of IEEE Std 1003.1-2001. The standard was approved by ISO and IEC and published in four parts, correlating to the original volumes.

A mapping of the parts to the volumes is shown below:

ISO/IEC 9945 Part	IEEE Std 1003.1 Volume	Description
9945-1	Base Definitions	Includes general terms, concepts, and interfaces common to all parts of ISO/IEC 9945, including utility conventions and C-language header definitions.
9945-2	System Interfaces	Includes definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery.
9945-3	Shell and Utilities	Includes definitions for a standard source code-level interface to command interpretation services (a ''shell'') and common utility programs for application programs.
9945-4	Rationale	Includes extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of ISO/IEC 9945 and why features were included or discarded by the standard developers.

All four parts comprise the entire standard, and are intended to be used together to accommodate significant internal referencing among them. POSIX-conforming systems are required to support all four parts.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

Note: This introduction is not part of IEEE Std 1003.1-2001, Standard for Information Technology — Portable Operating System Interface (POSIX).

This standard has been jointly developed by the IEEE and The Open Group. It is simultaneously an IEEE Standard, an ISO/IEC Standard, and an Open Group Technical Standard.

The Austin Group

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.[3] The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group has been to revise, combine, and update the following standards: ISO/IEC 9945-1, ISO/IEC 9945-2, IEEE Std 1003.1, IEEE Std 1003.2, and the Base Specifications of The Open Group Single UNIX Specification.

After two initial meetings, an agreement was signed in July 1999 between The Open Group and the Institute of Electrical and Electronics Engineers (IEEE), Inc., to formalize the project with the first draft of the revised specifications being made available at the same time. Under this agreement, The Open Group and IEEE agreed to share joint copyright of the resulting work. The Open Group has provided the chair and secretariat for the Austin Group.

The base document for the revision was The Open Group's Base volumes of its Single UNIX Specification, Version 2. These were selected since they were a superset of the existing POSIX.1 and POSIX.2 specifications and had some organizational aspects that would benefit the audience for the new revision.

The approach to specification development has been one of ''write once, adopt everywhere'', with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group's Technical Standard designation, and an ISO/IEC designation. This set of specifications forms the core of the Single UNIX Specification, Version 3.

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

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

---

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

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Background

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

Conceptually, this standard describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application writers to write portable applications—it was developed with that goal in mind—it has been designated POSIX,[4] an acronym for Portable Operating System Interface.

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

Audience

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

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

Purpose

Several principles guided the development of this standard:

- Application-Oriented

  The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. This standard codifies the common, existing definition of the UNIX system.

- Interface, Not Implementation

  This standard defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.

---

4. The name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

-`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- Source, Not Object, Portability

  This standard has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. This standard does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.

- The C Language

  The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.

- No Superuser, No System Administration

  There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in this standard. This standard is also not concerned with hardware constraints or system maintenance.

- Minimal Interface, Minimally Defined

  In keeping with the historical design principles of the UNIX system, the mandatory core facilities of this standard have been kept as minimal as possible. Additional capabilities have been added as optional extensions.

- Broadly Implementable

  The developers of this standard endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:

  1. All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
  2. Compatible systems that are not derived from the original UNIX system code
  3. Emulations hosted on entirely different operating systems
  4. Networked systems
  5. Distributed systems
  6. Systems running on a broad range of hardware

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

- Minimal Changes to Historical Implementations

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

  The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and the subsequent revisions and addenda to all of them have consolidated this consensus, and this revision reflects the significantly increased level of consensus arrived at since the original versions. The earlier standards and their modifications specified a number of areas where consensus had not been reached before, and these are now reflected in this revision. The authors of the original versions tried, as much as possible, to follow the principles below

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

when creating new specifications:

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

This revision tries to minimize the number of changes required to implementations which conform to the earlier versions of the approved standards to bring them into conformance with the current standard. Specifically, the scope of this work excluded doing any ''new'' work, but rather collecting into a single document what had been spread across a number of documents, and presenting it in what had been proven in practice to be a more effective way. Some changes to prior conforming implementations were unavoidable, primarily as a consequence of resolving conflicts found in prior revisions, or which became apparent when bringing the various pieces together.

However, since it references the 1999 version of the ISO C standard, and no longer supports ''Common Usage C'', there are a number of unavoidable changes. Applications portability is similarly affected.

This standard is specifically not a codification of a particular vendor's product.

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

- Minimal Changes to Existing Application Code

  A goal of this standard was to minimize additional work for the developers of applications. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

This Standard

This standard defines the Portable Operating System Interface (POSIX) requirements and consists of the following volumes:

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

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

This Volume

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

This volume is structured as follows:

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

Comprehensive references are available in the index.

Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as IEEE Std 1003.1-2001, which is technically identical to The Open Group Base Specifications, Issue 6.

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

Reference	Example	Notes
C-Language Data Structure	**aiocb**	
C-Language Data Structure Member	*aio_lio_opcode*	
C-Language Data Type	**long**	
C-Language External Variable	*errno*	
C-Language Function	*system*( )	
C-Language Function Argument	*arg1*	
C-Language Function Family	*exec*	
C-Language Header	**<sys/stat.h>**	
C-Language Keyword	**return**	
C-Language Macro with Argument	*assert*( )	
C-Language Macro with No Argument	INET_ADDRSTRLEN	
C-Language Preprocessing Directive	**#define**	
Commands within a Utility	**a**, **c**	
Conversion Specification, Specifier / Modifier Character	`%A`, `g`, `E`	1
Environment Variable	*PATH*	
Error Number	[EINTR]	
Example Output	**`Hello, World`**	
Filename	**/tmp**	
Literal Character	`'c'`,`'\r'`,`'\'`	2
Literal String	`"abcde"`	2
Optional Items in Utility Syntax	**[]**	
Parameter	<*directory pathname*>	
Special Character	<newline>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	`#include <sys/stat.h>`	

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Reference	Example	Notes
User Input and Example Code	`echo Hello, World`	5
Utility Name	*awk*	
Utility Operand	*file_name*	
Utility Option	**–c**	
Utility Option with Option-Argument	**–w** *width*	

Notes:

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf* and *fscanf* formatting functions.
2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. For literal characters, `'\'` (or any of the other sequences such as `'''`) is the same as the C constant `'\\'` (or `'\''`).
3. The style selected for some of the special characters, such as <newline>, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters <tab> or <newline>.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
5. Brackets shown in this font, `"[]"`, are part of the syntax and do *not* indicate optional items. In syntax the `'|'` symbol is used to separate alternatives, and ellipses (`"..."`) are used to show that additional arguments are optional.

Shading is used to identify extensions and options; see Section 1.8.1 (on page 3).

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

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

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

- (*a*,*b*) means the range of all values from *a* to *b*, including neither *a* nor *b*.
- [*a*,*b*] means the range of all values from *a* to *b*, including *a* and *b*.
- [*a*,*b*) means the range of all values from *a* to *b*, including *a*, but not *b*.
- (*a*,*b*] means the range of all values from *a* to *b*, including *b*, but not *a*.

Notes:

1. Symbolic limits are used in this volume instead of fixed values for portability. The values of most of these constants are defined in the Base Definitions volume, **<limits.h>** or **<unistd.h>**.
2. The values of errors are defined in the Base Definitions volume, **<errno.h>**.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Participants

IEEE Std 1003.1-2001 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/SC22 WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair
Donald W. Cragun, Organizational Representative, IEEE PASC
Nicholas Stoughton, Organizational Representative, ISO/SC22 WG15
Mark Brown, Organizational Representative, The Open Group
Cathy Hughes, Technical Editor

Austin Group Technical Reviewers

Peter Anvin
Bouazza Bachar
Theodore P. Baker
Walter Briscoe
Mark Brown
Dave Butenhof
Geoff Clare
Donald W. Cragun
Lee Damico
Ulrich Drepper
Paul Eggert
Joanna Farley
Clive D.W. Feather
Andrew Gollan
Michael Gonzalez
Joseph M. Gwinn
Jon Hitchcock
Yvette Ho Sang
Cathy Hughes
Lowell G. Johnson
Andrew Josey
Michael Kavanaugh
David Korn
Marc Aurele La France
Jim Meyering
Gary Miller
Finnbarr P. Murphy
Joseph S. Myers
Sandra O'Donnell
Frank Prindle
Curtis Royster Jr.
Glen Seeds
Keld Jorn Simonsen
Raja Srinivasan
Nicholas Stoughton
Donn S. Terry
Fred Tydeman
Peter Van Der Veen
James Youngman
Jim Zepeda
Jason Zions

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Austin Group Working Group Members

Harold C. Adams
Peter Anvin
Pierre-Jean Arcos
Jay Ashford
Bouazza Bachar
Theodore P. Baker
Robert Barned
Joel Berman
David J. Blackwood
Shirley Bockstahler-Brandt
James Bottomley
Walter Briscoe
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Andries Brouwer
Dave Butenhof
Keith Chow
Geoff Clare
Donald W. Cragun
Lee Damico
Juan Antonio De La Puente
Ming De Zhou
Steven J. Dovich
Richard P. Draves
Ulrich Drepper
Paul Eggert
Philip H. Enslow
Joanna Farley
Clive D.W. Feather
Pete Forman
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan
Michael Gonzalez
Karen D. Gordon
Joseph M. Gwinn
Steven A. Haaser
Charles E. Hammons
Chris J. Harding
Barry Hedquist
Vincent E. Henley
Karl Heubaum
Jon Hitchcock
Yvette Ho Sang
Niklas Holsti
Thomas Hosmer
Cathy Hughes
Jim D. Isaak
Lowell G. Johnson
Michael B. Jones
Andrew Josey
Michael J. Karels
Michael Kavanaugh
David Korn
Steven Kramer
Thomas M. Kurihara
Marc Aurele La France
C. Douglass Locke
Nick Maclaren
Roger J. Martin
Craig H. Meyer
Jim Meyering
Gary Miller
Finnbarr P. Murphy
Joseph S. Myers
John Napier
Peter E. Obermayer
James T. Oblinger
Sandra O'Donnell
Frank Prindle
Francois Riche
John D. Riley
Andrew K. Roach
Helmut Roth
Jaideep Roy
Curtis Royster Jr.
Stephen C. Schwarm
Glen Seeds
Richard Seibel
David L. Shroads Jr.
W. Olin Sibert
Keld Jorn Simonsen
Curtis Smith
Raja Srinivasan
Nicholas Stoughton
Marc J. Teller
Donn S. Terry
Fred Tydeman
Mark-Rene Uchida
Scott A. Valcourt
Peter Van Der Veen
Michael W. Vannier
Eric Vought
Frederick N. Webb
Paul A.T. Wolfgang
Garrett A. Wollman
James Youngman
Oren Yuen
Janusz Zalewski
Jim Zepeda
Jason Zions

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The Open Group

When The Open Group approved the Base Specifications, Issue 6 on 12 September 2001, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair
Finnbarr P. Murphy, Vice-Chair
Mark Brown, Austin Group Liaison
Cathy Hughes, Technical Editor

Base Working Group Members

Bouazza Bachar
Mark Brown
Dave Butenhof
Donald W. Cragun
Larry Dwyer
Joanna Farley
Andrew Gollan
Karen D. Gordon
Gary Miller
Finnbarr P. Murphy
Frank Prindle
Andrew K. Roach
Curtis Royster Jr.
Nicholas Stoughton
Kenjiro Tsuji

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair
Joseph M. Gwinn, Vice-Chair
Jay Ashford, Functional Chair
Andrew Josey, Functional Chair
Curtis Royster Jr., Functional Chair
Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001. Balloters may have voted for approval, disapproval, or abstention:

Harold C. Adams
Pierre-Jean Arcos
Jay Ashford
Theodore P. Baker
Robert Barned
David J. Blackwood
Shirley Bockstahler-Brandt
James Bottomley
Mark Brown
Eric W. Burger
Alan Burns
Dave Butenhof
Keith Chow
Donald W. Cragun
Juan Antonio De La Puente
Ming De Zhou
Steven J. Dovich
Richard P. Draves
Philip H. Enslow
Michael Gonzalez
Karen D. Gordon
Joseph M. Gwinn
Steven A. Haaser
Charles E. Hammons
Chris J. Harding
Barry Hedquist
Vincent E. Henley
Karl Heubaum
Niklas Holsti
Thomas Hosmer
Jim D. Isaak
Lowell G. Johnson
Michael B. Jones
Andrew Josey
Michael J. Karels
Steven Kramer
Thomas M. Kurihara
C. Douglass Locke
Roger J. Martin
Craig H. Meyer
Finnbarr P. Murphy
John Napier
Peter E. Obermayer
James T. Oblinger
Frank Prindle
Francois Riche
John D. Riley
Andrew K. Roach
Helmut Roth
Jaideep Roy
Curtis Royster Jr.
Stephen C. Schwarm
Richard Seibel
David L. Shroads Jr.
W. Olin Sibert
Keld Jorn Simonsen
Nicholas Stoughton
Donn S. Terry
Mark-Rene Uchida
Scott A. Valcourt
Michael W. Vannier
Frederick N. Webb
Paul A.T. Wolfgang
Oren Yuen
Janusz Zalewski

The following organizational representative voted on this standard:

Andrew Josey, X/Open Company Ltd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, it had the following membership:

Donald N. Heirman, Chair
James T. Carlo, Vice-Chair
Judith Gorman, Secretary

Satish K. Aggarwal	James H. Gurney	James W. Moore
Mark D. Bowman	Richard J. Holleman	Robert F. Munzner
Gary R. Engmann	Lowell G. Johnson	Ronald C. Petersen
Harold E. Epstein	Robert J. Kennelly	Gerald H. Peterson
H. Landis Floyd	Joseph L. Koepfinger*	John B. Posey
Jay Forster*	Peter H. Lips	Gary S. Robinson
Howard M. Frazier	L. Bruce McClung	Akio Tojo
Ruben D. Garzon	Daleep C. Mohla	Donald W. Zipse

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative
Donald R. Volzka, TAB Representative
Yvette Ho Sang, **Don Messina**, **Savoula Amanatidis**, IEEE Project Editors

* Member Emeritus

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE Std 1003.1-2001/Cor 1-2002 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22/WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair
Donald W. Cragun, Organizational Representative, IEEE PASC
Nicholas Stoughton, Organizational Representative, ISO/IEC JTC 1/SC22/WG15
Mark Brown, Organizational Representative, The Open Group
Cathy Fox, Technical Editor

Austin Group Technical Reviewers

Theodore P. Baker
Julian Blake
Andries Brouwer
Mark Brown
Dave Butenhof
Geoff Clare
Donald W. Cragun
Ken Dawson
Ulrich Drepper
Larry Dwyer
Paul Eggert
Joanna Farley
Clive D.W. Feather
Cathy Fox
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan
Michael Gonzalez
Bruno Haible
Ben Harris
Jon Hitchcock
Andreas Jaeger
Andrew Josey
Jonathan Lennox
Nick Maclaren
Jack McCann
Wilhelm Mueller
Joseph S. Myers
Frank Prindle
Kenneth Raeburn
Tim Robbins
Glen Seeds
Matthew Seitz
Keld Jorn Simonsen
Nicholas Stoughton
Alexander Terekhov
Donn S. Terry
Mike Wilson
Garrett A. Wollman
Mark Ziegast

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Austin Group Working Group Members

Harold C. Adams
Alejandro Alonso
Jay Ashford
Theodore P. Baker
David J. Blackwood
Julian Blake
Mitchell Bonnett
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Dave Butenhof
Keith Chow
Geoff Clare
Luis Cordova
Donald W. Cragun
Dragan Cvetkovic
Lee Damico
Ken Dawson
Jeroen Dekkers
Juan Antonio De La Puente
Steven J. Dovich
Ulrich Drepper
Dr. Sourav Dutta
Larry Dwyer
Paul Eggert
Joanna Farley
Clive D.W. Feather
Yaacov Fenster
Cathy Fox
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan
Michael Gonzalez
Karen D. Gordon
Scott Gudgel
Joseph M. Gwinn
Steven A. Haaser
Bruno Haible
Charles E. Hammons
Bryan Harold
Ben Harris
Barry Hedquist
Karl Heubaum
Jon Hitchcock
Andreas Jaeger
Andrew Josey
Kenneth Lang
Pi-Cheng Law
Jonathan Lennox
Nick Maclaren
Roger J. Martin
Jack McCann
George Miao
Wilhelm Mueller
Finnbarr P. Murphy
Joseph S. Myers
Alexey Neyman
Charles Ngethe
Peter Petrov
Frank Prindle
Vikram Punj
Kenneth Raeburn
Francois Riche
Tim Robbins
Curtis Royster Jr.
Diane Schleicher
Gil Shultz
Stephen C. Schwarm
Glen Seeds
Matthew Seitz
Keld Jorn Simonsen
Doug Stevenson
Nicholas Stoughton
Alexander Terekhov
Donn S. Terry
Mike Wilson
Garrett A. Wollman
Oren Yuen
Mark Ziegast

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The Open Group

When The Open Group approved the Base Specifications, Issue 6, Technical Corrigendum 1 on 7 February 2003, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair
Finnbarr P. Murphy, Vice-Chair
Mark Brown, Austin Group Liaison
Cathy Fox, Technical Editor

Base Working Group Members

Mark Brown
Dave Butenhof
Donald W. Cragun
Larry Dwyer
Ulrich Drepper
Joanna Farley
Andrew Gollan
Finnbarr P. Murphy
Frank Prindle
Andrew K. Roach
Curtis Royster Jr.
Nicholas Stoughton
Kenjiro Tsuji

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair
Joseph M. Gwinn, Vice-Chair
Jay Ashford, Functional Chair
Andrew Josey, Functional Chair
Curtis Royster Jr., Functional Chair
Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001/Cor 1-2002. Balloters may have voted for approval, disapproval, or abstention:

Alejandro Alonso
Jay Ashford
David J. Blackwood
Julian Blake
Mitchell Bonnett
Mark Brown
Dave Butenhof
Keith Chow
Luis Cordova
Donald W. Cragun
Steven J. Dovich
Dr. Sourav Dutta
Yaacov Fenster
Michael Gonzalez
Scott Gudgel
Charles E. Hammons
Bryan Harold
Barry Hedquist
Karl Heubaum
Lowell G. Johnson
Andrew Josey
Kenneth Lang
Pi-Cheng Law
George Miao
Roger J. Martin
Finnbarr P. Murphy
Charles Ngethe
Peter Petrov
Frank Prindle
Vikram Punj
Francois Riche
Curtis Royster Jr.
Diane Schleicher
Stephen C. Schwarm
Gil Shultz
Nicholas Stoughton
Donn S. Terry
Oren Yuen
Juan A. de la Puente

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership was as follows:

James T. Carlo, Chair
James H. Gurney, Vice-Chair
Judith Gorman, Secretary

Sid Bennett	Arnold M. Greenspan	Daleep C. Mohla
H. Stephen Berger	Raymond Hapeman	William J. Moylan
Clyde R. Camp	Donald M. Heirman	Malcolm V. Thaden
Richard DeBlasio	Richard H. Hulett	Geoffrey O. Thompson
Harold E. Epstein	Lowell G. Johnson	Howard L. Wolfman
Julian Forster*	Joseph L. Koepfinger*	Don Wright
Howard M. Frazier	Peter H. Lips	
Toshio Fukuda	Nader Mehravari	

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative
Satish K. Aggarwal, NRC Representative
Savoula Amanatidis, IEEE Standards Managing Editor

* Member Emeritus

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Trademarks

The following information is given for the convenience of users of this standard and does not constitute endorsement of these products by The Open Group or the IEEE. There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

1003.1™ is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

AIX® is a registered trademark of IBM Corporation.

AT&T® is a registered trademark of AT&T in the U.S.A. and other countries.

BSD™ is a trademark of the University of California, Berkeley, U.S.A.

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

IBM® is a registered trademark of International Business Machines Corporation.

The Open Group and Boundaryless Information Flow are trademarks and UNIX is a registered trademark of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

POSIX® is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

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

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

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Acknowledgements

The contributions of the following organizations to the development of IEEE Std 1003.1-2001 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- The SC22 WG14 Committees.

This standard was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO SC22 WG15.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Referenced Documents

Normative References

Normative references for this standard are defined in the Base Definitions volume.

Informative References

The following documents are referenced in this standard:

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

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

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

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

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

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

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

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

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

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

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

IEEE Std 754-1985
IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.

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

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

IETF RFC 791
Internet Protocol, Version 4 (IPv4), September 1981.

IETF RFC 819
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982.

IETF RFC 822
Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982.

IETF RFC 919
Broadcasting Internet Datagrams, J. Mogul, October 1984.

IETF RFC 920
Domain Requirements, J. Postel, J. Reynolds, October 1984.

IETF RFC 921
Domain Name System Implementation Schedule, J. Postel, October 1984.

IETF RFC 922
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984.

IETF RFC 1034
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987.

IETF RFC 1035
Domain Names — Implementation and Specification, P. Mockapetris, November 1987.

IETF RFC 1123
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989.

IETF RFC 1886
DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995.

IETF RFC 2045
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996.

IETF RFC 2181
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997.

IETF RFC 2373
Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998.

IETF RFC 2460
Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998.

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

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

ISO / IEC 1539: 1990
ISO / IEC 1539: 1990, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

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

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

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

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

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

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

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

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

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 1

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

Issue 2

X/Open Portability Guide, January 1987:

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

Issue 3

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

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

Issue 4

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

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

Issue 4, Version 2

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

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

Issue 5

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

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

Knuth Article

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

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

KornShell

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

MSE Working Draft

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

POSIX.0:1995

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

POSIX.1:1988

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

POSIX.1:1990

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

POSIX.1a

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

POSIX.1d:1999

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

POSIX.1g:2000

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

POSIX.1j:2000

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

POSIX.1q:2000

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

POSIX.2b

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

POSIX.2d:-1994

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

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

POSIX.13:-1998

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

Sarwate Article

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

Sprunt, Sha, and Lehoczky

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

SVID, Issue 1

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

SVID, Issue 2

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

SVID, Issue 3

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

The AWK Programming Language

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

UNIX Programmer's Manual

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

XNS, Issue 4

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

XNS, Issue 5

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

XNS, Issue 5.2

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

X/Open Curses, Issue 4, Version 2

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

Yacc

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

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Source Documents

Parts of the following documents were used to create the base documents for this standard:

AIX 3.2 Manual

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

OSF/1

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

OSF AES

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

System V Release 2.0

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

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

System V Release 4.2

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

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Chapter 1

Introduction

1.1 Scope

The scope of IEEE Std 1003.1-2001 is described in the Base Definitions volume of IEEE Std 1003.1-2001.

1.2 Conformance

Conformance requirements for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance.

1.3 Normative References

Normative references for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

1.4 Change History

Change history is described in the Rationale (Informative) volume of IEEE Std 1003.1-2001, and in the CHANGE HISTORY section of reference pages.

1.5 Terminology

This section appears in the Base Definitions volume of IEEE Std 1003.1-2001, but is repeated here for convenience:

For the purposes of IEEE Std 1003.1-2001, the following terminology definitions apply:

can

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to IEEE Std 1003.1-2001. An application can rely on the existence of the feature or behavior.

implementation-defined

Describes a value or behavior that is not defined by IEEE Std 1003.1-2001 but is selected by an implementor. The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

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

legacy

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

may

Describes a feature or behavior that is optional for an implementation that conforms to IEEE Std 1003.1-2001. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

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

shall

For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or behavior that is mandatory. An application can rely on the existence of the feature or behavior.

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

should

For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

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

undefined

Describes the nature of a value or behavior not defined by IEEE Std 1003.1-2001 which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

unspecified

Describes the nature of a value or behavior not specified by IEEE Std 1003.1-2001 which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6 Definitions

Concepts and definitions are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

1.7 Relationship to Other Formal Standards

Great care has been taken to ensure that this volume of IEEE Std 1003.1-2001 is fully aligned with the following standards:

ISO C (1999)
ISO/IEC 9899:1999, Programming Languages — C.

Parts of the ISO/IEC 9899:1999 standard (hereinafter referred to as the ISO C standard) are referenced to describe requirements also mandated by this volume of IEEE Std 1003.1-2001. Some functions and headers included within this volume of IEEE Std 1003.1-2001 have a version in the ISO C standard; in this case CX markings are added as appropriate to show where the ISO C standard has been extended (see Section 1.8.1). Any conflict between this volume of IEEE Std 1003.1-2001 and the ISO C standard is unintentional.

This volume of IEEE Std 1003.1-2001 also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1987.

1.8 Portability

Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 and functions in the System Interfaces volume of IEEE Std 1003.1-2001 describe functionality that might not be fully portable to systems meeting the requirements for POSIX conformance (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance).

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

1.8.1 Codes

Margin codes and their meanings are listed in the Base Definitions volume of IEEE Std 1003.1-2001, but are repeated here for convenience:

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

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

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

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

BAR Barriers
The functionality described is optional. The functionality described is also an extension to the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ISO C standard.

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

BE Batch Environment Services and Utilities
The functionality described is optional.

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

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

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

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

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

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

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

CX Extension to the ISO C standard
The functionality described is an extension to the ISO C standard. Application writers may make use of an extension as it is supported on all IEEE Std 1003.1-2001-conforming systems.

With each function or header from the ISO C standard, a statement to the effect that ‘‘any conflict is unintentional’’ is included. That is intended to refer to a direct conflict. IEEE Std 1003.1-2001 acts in part as a profile of the ISO C standard, and it may choose to further constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered conflicts.

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

FD FORTRAN Development Utilities
The functionality described is optional.

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

FR FORTRAN Runtime Utilities
The functionality described is optional.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

FSC File Synchronization

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

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

IP6 IPV6

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

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

MC1 Advisory Information and either Memory Mapped Files or Shared Memory Objects

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

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

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

Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code Notation.

MC2 Memory Mapped Files, Shared Memory Objects, or Memory Protection

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

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

Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MC2 margin legend.

Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code Notation.

MC3 Memory Mapped Files, Shared Memory Objects, or Typed Memory Objects

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

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

Where applicable, functions are marked with the MC3 margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MC3 margin legend.

Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code Notation.

MF Memory Mapped Files

The functionality described is optional. The functionality described is also an extension to the

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ISO C standard.

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

ML Process Memory Locking

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

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

MLR Range Memory Locking

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

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

MON Monotonic Clock

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

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

MPR Memory Protection

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

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

MSG Message Passing

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

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

MX IEC 60559 Floating-Point Option

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

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

OB Obsolescent

The functionality described may be withdrawn in a future version of this volume of IEEE Std 1003.1-2001. Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use obsolescent features.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

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

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

OH **Optional Header**
In the SYNOPSIS section of some interfaces in the System Interfaces volume of IEEE Std 1003.1-2001 an included header is marked as in the following example:

OH
```
#include <sys/types.h>
#include <grp.h>
struct group *getgrnam(const char *name);
```

The OH margin legend indicates that the marked header is not required on XSI-conformant systems.

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

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

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

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

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

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

RTS **Realtime Signals Extension**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

SD **Software Development Utilities**
The functionality described is optional.

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

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

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

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

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

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

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

SPI **Spin Locks**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

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

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

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

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

TCT **Thread CPU-Time Clocks**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

TEF **Trace Event Filter**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

THR Threads

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

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

TMO Timeouts

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

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

TMR Timers

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

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

TPI Thread Priority Inheritance

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

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

TPP Thread Priority Protection

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

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

TPS Thread Execution Scheduling

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

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

TRC Trace

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

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

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

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

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

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

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

TSF Thread-Safe Functions
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

TSH Thread Process-Shared Synchronization
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

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

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

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

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

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

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

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

UP User Portability Utilities
The functionality described is optional.

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

XSI Extension
The functionality described is an XSI extension. Functionality marked XSI is also an extension to the ISO C standard. Application writers may confidently make use of an extension on all systems supporting the X/Open System Interfaces Extension.

If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that reference page is an extension. See the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.439, XSI.

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

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

1.9 Format of Entries

The entries in Chapter 3 are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described. If it is necessary to include a header to use this function, the names of such headers are shown, for example:

```
#include <stdio.h>
```

DESCRIPTION

This section describes the functionality of the function or header.

RETURN VALUE

This section indicates the possible return values, if any.

If the implementation can detect errors, ‘‘successful completion’’ means that no error has been detected during execution of the function. If the implementation does detect an error, the error is indicated.

For functions where no errors are defined, ‘‘successful completion’’ means that if the implementation checks for errors, no error has been detected. If the implementation can detect errors, and an error is detected, the indicated return value is returned and *errno* may be set.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ERRORS

This section gives the symbolic names of the error values returned by a function or stored into a variable accessed through the symbol *errno* if an error occurs.

''No errors are defined'' means that error values returned by a function or stored into a variable accessed through the symbol *errno*, if any, depend on the implementation.

EXAMPLES

This section is informative.

This section gives examples of usage, where appropriate. In the event of conflict between an example and a normative part of this volume of IEEE Std 1003.1-2001, the normative material is to be taken as correct.

APPLICATION USAGE

This section is informative.

This section gives warnings and advice to application writers about the entry. In the event of conflict between warnings and advice and a normative part of this volume of IEEE Std 1003.1-2001, the normative material is to be taken as correct.

RATIONALE

This section is informative.

This section contains historical information concerning the contents of this volume of IEEE Std 1003.1-2001 and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is informative.

This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO

This section is informative.

This section gives references to related information.

CHANGE HISTORY

This section is informative.

This section shows the derivation of the entry and any significant changes that have been made to it.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Chapter 2

General Information

This chapter covers information that is relevant to all the functions specified in Chapter 3 and the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

2.1 Use and Implementation of Functions

Each of the following statements shall apply unless explicitly stated otherwise in the detailed descriptions that follow:

1. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behavior is undefined.

2. Any function declared in a header may also be implemented as a macro defined in the header, so a function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a function even if it is also defined as a macro. The use of the C-language **#undef** construct to remove any such macro definition shall also ensure that an actual function is referred to.

3. Any invocation of a function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression anywhere a function with a compatible return type could be called.

4. Provided that a function can be declared without reference to any type defined in a header, it is also permissible to declare the function explicitly and use it without including its associated header.

5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

2.2 The Compilation Environment

2.2.1 POSIX.1 Symbols

Certain symbols in this volume of IEEE Std 1003.1-2001 are defined in headers (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers). Some of those headers could also define symbols other than those defined by IEEE Std 1003.1-2001, potentially conflicting with symbols used by the application. Also, IEEE Std 1003.1-2001 defines symbols that are not permitted by other standards to appear in those headers without some control on the visibility of those symbols.

Symbols called ‘‘feature test macros’’ are used to control the visibility of symbols that might be included in a header. Implementations, future versions of IEEE Std 1003.1-2001, and other standards may define additional feature test macros.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

In the compilation of an application that **#define**s a feature test macro specified by IEEE Std 1003.1-2001, no header defined by IEEE Std 1003.1-2001 shall be included prior to the definition of the feature test macro. This restriction also applies to any implementation-provided header in which these feature test macros are used. If the definition of the macro does not precede the **#include**, the result is undefined.

Feature test macros shall begin with the underscore character ('_').

2.2.1.1 The _POSIX_C_SOURCE Feature Test Macro

A POSIX-conforming application should ensure that the feature test macro _POSIX_C_SOURCE is defined before inclusion of any header.

When an application includes a header described by IEEE Std 1003.1-2001, and when this feature test macro is defined to have the value 200112L:

1. All symbols required by IEEE Std 1003.1-2001 to appear when the header is included shall be made visible.
2. Symbols that are explicitly permitted, but not required, by IEEE Std 1003.1-2001 to appear in that header (including those in reserved name spaces) may be made visible.
3. Additional symbols not required or explicitly permitted by IEEE Std 1003.1-2001 to be in that header shall not be made visible, except when enabled by another feature test macro.

Identifiers in IEEE Std 1003.1-2001 may only be undefined using the **#undef** directive as described in Section 2.1 (on page 13) or Section 2.2.2. These **#undef** directives shall follow all **#include** directives of any header in IEEE Std 1003.1-2001.

Note: The POSIX.1-1990 standard specified a macro called _POSIX_SOURCE. This has been superseded by _POSIX_C_SOURCE.

2.2.1.2 The _XOPEN_SOURCE Feature Test Macro

XSI An XSI-conforming application should ensure that the feature test macro _XOPEN_SOURCE is defined with the value 600 before inclusion of any header. This is needed to enable the functionality described in Section 2.2.1.1 and in addition to enable the XSI extension.

Since this volume of IEEE Std 1003.1-2001 is aligned with the ISO C standard, and since all functionality enabled by _POSIX_C_SOURCE set equal to 200112L is enabled by _XOPEN_SOURCE set equal to 600, there should be no need to define _POSIX_C_SOURCE if _XOPEN_SOURCE is so defined. Therefore, if _XOPEN_SOURCE is set equal to 600 and _POSIX_C_SOURCE is set equal to 200112L, the behavior is the same as if only _XOPEN_SOURCE is defined and set equal to 600. However, should _POSIX_C_SOURCE be set to a value greater than 200112L, the behavior is unspecified.

2.2.2 The Name Space

All identifiers in this volume of IEEE Std 1003.1-2001, except *environ*, are defined in at least one of the headers, as shown in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers. When XSI _XOPEN_SOURCE or _POSIX_C_SOURCE is defined, each header defines or declares some identifiers, potentially conflicting with identifiers used by the application. The set of identifiers visible to the application consists of precisely those identifiers from the header pages of the included headers, as well as additional identifiers reserved for the implementation. In addition, some headers may make visible identifiers from other headers as indicated on the relevant header pages.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Implementations may also add members to a structure or union without controlling the visibility of those members with a feature test macro, as long as a user-defined macro with the same name cannot interfere with the correct interpretation of the program. The identifiers reserved for use by the implementation are described below:

1. Each identifier with external linkage described in the header section is reserved for use as an identifier with external linkage if the header is included.
2. Each macro described in the header section is reserved for any use if the header is included.
3. Each identifier with file scope described in the header section is reserved for use as an identifier with file scope in the same name space if the header is included.

The prefixes posix_, POSIX_, and _POSIX_ are reserved for use by IEEE Std 1003.1-2001 and other POSIX standards. Implementations may add symbols to the headers shown in the following table, provided the identifiers for those symbols begin with the corresponding reserved prefixes in the following table, and do not use the reserved prefixes posix_, POSIX_, or _POSIX_.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

	Header	Prefix	Suffix	Complete Name
AIO	**<aio.h>**	aio_, lio_, AIO_, LIO_		
	<arpa/inet.h>	in_, inet_		
	<ctype.h>	to[a-z], is[a-z]		
	<dirent.h>	d_		
	<errno.h>	E[0-9], E[A-Z]		
	<fcntl.h>	l_		
	<glob.h>	gl_		
	<grp.h>	gr_		
	<inttypes.h>			int[0-9a-z_]*_t, uint[0-9a-z_]*_t
	<limits.h>		_MAX, _MIN	
	<locale.h>	LC_[A-Z]		
MSG	**<mqueue.h>**	mq_, MQ_		
XSI	**<ndbm.h>**	dbm_		
	<netdb.h>	h_, n_, p_, s_		
	<net/if.h>	if_		
	<netinet/in.h>	in_, ip_, s_, sin_		
IP6		in6_, s6_, sin6_		
XSI	**<poll.h>**	pd_, ph_, ps_		
	<pthread.h>	pthread_, PTHREAD_		
	<pwd.h>	pw_		
	<regex.h>	re_, rm_		
PS	**<sched.h>**	sched_, SCHED_		
SEM	**<semaphore.h>**	sem_, SEM_		
	<signal.h>	sa_, uc_, SIG[A-Z], SIG_[A-Z]		
XSI		ss_, sv_		
RTS		si_, SI_, sigev_, SIGEV_, sival_		
XSI	**<stropts.h>**	bi_, ic_, l_, sl_, str_		
	<stdint.h>			int[0-9a-z_]*_t, uint[0-9a-z_]*_t
	<stdlib.h>	str[a-z]		
	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
XSI	**<sys/ipc.h>**	ipc_		key, pad, seq
MF	**<sys/mman.h>**	shm_, MAP_, MCL_, MS_, PROT_		
XSI	**<sys/msg.h>**	msg		msg
XSI	**<sys/resource.h>**	rlim_, ru_		
	<sys/select.h>	fd_, fds_, FD_		
XSI	**<sys/sem.h>**	sem		sem
XSI	**<sys/shm.h>**	shm		
	<sys/socket.h>	ss_, sa_, if_, ifc_, ifru_, infu_, ifra_, msg_, cmsg_, l_		
	<sys/stat.h>	st_		
XSI	**<sys/statvfs.h>**	f_		
	<sys/time.h>	fds_, it_, tv_, FD_		
	<sys/times.h>	tms_		

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	Header	Prefix	Suffix	Complete Name
XSI	**<sys/uio.h>**	iov_		UIO_MAXIOV
	<sys/un.h>	sun_		
	<sys/utsname.h>	uts_		
XSI	**<sys/wait.h>**	si_, W[A-Z], P_		
	<termios.h>	c_		
	<time.h>	tm_		
TMR		clock_, timer_, it_, tv_,		
TMR		CLOCK_, TIMER_		
XSI	**<ucontext.h>**	uc_, ss_		
XSI	**<ulimit.h>**	UL_		
	<utime.h>	utim_		
XSI	**<utmpx.h>**	ut_	_LVL, _TIME, _PROCESS	
	<wchar.h>	wcs[a-z]		
	<wctype.h>	is[a-z], to[a-z]		
	<wordexp.h>	we_		
	ANY header		_t	

Note: The notation [A–Z] indicates any uppercase letter in the portable character set. The notation [a–z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.

If any header in the following table is included, macros with the prefixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the corresponding macro.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

	Header	Prefix
XSI	**<dlfcn.h>**	RTLD_
	<fcntl.h>	F_, O_, S_
XSI	**<fmtmsg.h>**	MM_
	<fnmatch.h>	FNM_
XSI	**<ftw.h>**	FTW
	<glob.h>	GLOB_
	<inttypes.h>	PRI[Xa-z], SCN[Xa-z]
XSI	**<ndbm.h>**	DBM_
	<net/if.h>	IF_
	<netinet/in.h>	IMPLINK_, IN_, INADDR_, IP_, IPPORT_, IPPROTO_, SOCK_
IP6		IPV6_, IN6_
	<netinet/tcp.h>	TCP_
XSI	**<nl_types.h>**	NL_
XSI	**<poll.h>**	POLL
	<regex.h>	REG_
	<signal.h>	SA_, SIG_[0-9a-z_],
XSI		BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, SS_, SV_, TRAP_
XSI	**<stropts.h>**	FLUSH[A-Z], I_, M_, MUXID_R[A-Z], S_, SND[A-Z], STR
XSI	**<syslog.h>**	LOG_
XSI	**<sys/ipc.h>**	IPC_
XSI	**<sys/mman.h>**	PROT_, MAP_, MS_
XSI	**<sys/msg.h>**	MSG[A-Z]
XSI	**<sys/resource.h>**	PRIO_, RLIM_, RLIMIT_, RUSAGE_
XSI	**<sys/sem.h>**	SEM_
XSI	**<sys/shm.h>**	SHM[A-Z], SHM_[A-Z]
XSI	**<sys/socket.h>**	AF_, CMSG_, MSG_, PF_, SCM_, SHUT_, SO
	<sys/stat.h>	S_
XSI	**<sys/statvfs.h>**	ST_
XSI	**<sys/time.h>**	FD_, ITIMER_
XSI	**<sys/uio.h>**	IOV_
XSI	**<sys/wait.h>**	BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, TRAP_
	<termios.h>	V, I, O, TC, B[0-9] (See below.)
	<wordexp.h>	WRDE_

The following are used to reserve complete names for the **<stdint.h>** header:

INT[0-9A-Za-z_]*_MIN
INT[0-9A-Za-z_]*_MAX
INT[0-9A-Za-z_]*_C
UINT[0-9A-Za-z_]*_MIN
UINT[0-9A-Za-z_]*_MAX
UINT[0-9A-Za-z_]*_C

Note: The notation [0–9] indicates any digit. The notation [A–Z] indicates any uppercase letter in the portable character set. The notation [0–9a–z_] indicates any digit, any lowercase letter in the portable character set, or underscore.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The following reserved names are used as exact matches for **<termios.h>**:

XSI

CBAUD	EXTB	VDSUSP
DEFECHO	FLUSHO	VLNEXT
ECHOCTL	LOBLK	VREPRINT
ECHOKE	PENDIN	VSTATUS
ECHOPRT	SWTCH	VWERASE
EXTA	VDISCARD	

The following identifiers are reserved regardless of the inclusion of headers:

1. All identifiers that begin with an underscore and either an uppercase letter or another underscore are always reserved for any use by the implementation.
2. All identifiers that begin with an underscore are always reserved for use as identifiers with file scope in both the ordinary identifier and tag name spaces.
3. All identifiers in the table below are reserved for use as identifiers with external linkage. Some of these identifiers do not appear in this volume of IEEE Std 1003.1-2001, but are reserved for future use by the ISO C standard.

_Exit	ccoshf	csqrt	fputc	lrintl	sinh
abort	ccoshl	csqrtf	fputs	lround	sinhf
abs	ccosl	csqrtl	fputwc	lroundf	sinhl
acos	ceil	ctan	fputws	lroundl	sinl
acosf	ceilf	ctanf	fread	malloc	sprintf
acosh	ceilf	ctanl	free	mblen	sqrt
acoshf	ceill	ctgamma	freopen	mbrlen	sqrtf
acoshl	ceill	ctgammaf	frexp	mbrtowc	sqrtl
acosl	cerf	ctgammal	frexpf	mbsinit	srand
acosl	cerfc	ltime	frexpl	mbsrtowcs	sscanf
asctime	cerfcf	difftime	fscanf	mbstowcs	str[a-z]*
asin	cerfcl	div	fseek	mbtowc	strtof
asinf	cerff	erfcf	fsetpos	mem[a-z]*	strtoimax
asinh	cerfl	erfcl	ftell	mktime	strtold
asinhf	cexmp1	erff	fwide	modf	strtoll
asinhl	cexmp1f	erfl	fwprintf	modff	strtoull
asinl	cexmp1l	errno	fwrite	modfl	strtoumax
asinl	cexp	exit	fwscanf	nan	swprintf
atan	cexp2	exp	getc	nanf	swscanf
atan2	cexp2f	exp2	getchar	nanl	system
atan2f	cexp2l	exp2f	getenv	nearbyint	tan
atan2l	cexpf	exp2l	gets	nearbyintf	tanf
atanf	cexpl	expf	getwc	nearbyintl	tanh
atanf	cimag	expl	getwchar	nextafterf	tanhf
atanh	cimagf	expm1	gmtime	nextafterl	tanhl
atanh	cimagl	expm1f	hypotf	nexttoward	tanl
atanhf	clearerr	expm1l	hypotl	nexttowardf	tgamma
atanhl	clgamma	fabs	ilogb	nexttowardl	tgammaf
atanl	clgammaf	fabsf	ilogbf	perror	tgammal
atanl	clgammal	fabsl	ilogbl	pow	time
atexit	clock	fclose	imaxabs	powf	tmpfile

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

atof	clog	fdim	imaxdiv	powl	tmpnam
atoi	clog10	fdimf	is[a-z]*	printf	to[a-z]*
atol	clog10f	fdiml	isblank	putc	trunc
atoll	clog10l	feclearexcept	iswblank	putchar	truncf
bsearch	clog1p	fegetenv	labs	puts	truncl
cabs	clog1pf	fegetexceptflag	ldexp	putwc	ungetc
cabsf	clog1pl	fegetround	ldexpf	putwchar	ungetwc
cabsl	clog2	feholdexcept	ldexpl	qsort	va_end
cacos	clog2f	feof	ldiv	raise	vfprintf
cacosf	clog2l	feraiseexcept	ldiv	rand	vfscanf
cacosh	clogf	ferror	lgammaf	realloc	vfwprintf
cacoshf	clogl	fesetenv	lgammal	remainderf	vfwscanf
cacoshl	conj	fesetexceptflag	llabs	remainderl	vprintf
cacosl	conjf	fesetround	llrint	remove	vscanf
calloc	conjl	fetestexcept	llrintf	remquo	vsprintf
carg	copysign	feupdateenv	llrintl	remquof	vsscanf
cargf	copysignf	fflush	llround	remquol	vswprintf
cargl	copysignl	fgetc	llroundf	rename	vswscanf
casin	cos	fgetpos	llroundl	rewind	vwprintf
casinf	cosf	fgets	localeconv	rint	vwscanf
casinh	cosh	fgetwc	localtime	rintf	wcrtomb
casinhf	coshf	fgetws	log	rintl	wcs[a-z]*
casinhl	coshl	floor	log10	round	wcstof
casinl	cosl	floorf	log10f	roundf	wcstoimax
catan	cpow	floorl	log10l	roundl	wcstold
catanf	cpowf	fma	log1p	scalbln	wcstoll
catanh	cpowl	fmaf	log1pf	scalblnf	wcstoull
catanh	cproj	fmal	log1pl	scalblnl	wcstoumax
catanhf	cprojf	fmax	log2	scalbn	wctob
catanhf	cprojl	fmaxf	log2f	scalbnf	wctomb
catanhl	creal	fmaxl	log2l	scalbnl	wctrans
catanhl	crealf	fmin	logb	scanf	wctype
catanl	creall	fminf	logbf	setbuf	wcwidth
cbrt	csin	fminl	logbl	setjmp	wmem[a-z]*
cbrtf	csinf	fmod	logf	setlocale	wprintf
cbrtl	csinh	fmodf	logl	setvbuf	wscanf
ccos	csinhf	fmodl	longjmp	signal	
ccosf	csinhl	fopen	lrint	sin	
ccosh	csinl	fprintf	lrintf	sinf	

Note: The notation [a–z] indicates any lowercase letter in the portable character set. The notation '*' indicates any combination of digits, letters in the portable character set, or underscore.

4. All functions and external identifiers defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers are reserved for use as identifiers with external linkage.

5. All the identifiers defined in this volume of IEEE Std 1003.1-2001 that have external linkage are always reserved for use as identifiers with external linkage.

No other identifiers are reserved.

Applications shall not declare or define identifiers with the same name as an identifier reserved in the same context. Since macro names are replaced whenever found, independent of scope and

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

name space, macro names matching any of the reserved identifier names shall not be defined by an application if any associated header is included.

Except that the effect of each inclusion of **<assert.h>** depends on the definition of NDEBUG, headers may be included in any order, and each may be included more than once in a given scope, with no difference in effect from that of being included only once.

If used, the application shall ensure that a header is included outside of any external declaration or definition, and it shall be first included before the first reference to any type or macro it defines, or to any function or object it declares. However, if an identifier is declared or defined in more than one header, the second and subsequent associated headers may be included after the initial reference to the identifier. Prior to the inclusion of a header, the application shall not define any macros with names lexically identical to symbols defined by that header.

2.3 Error Numbers

Most functions can provide an error number. The means by which each function provides its error numbers is specified in its description.

Some functions provide the error number in a variable accessed through the symbol *errno*. The symbol *errno*, defined by including the **<errno.h>** header, expands to a modifiable lvalue of type **int**. It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

The value of *errno* should only be examined when it is indicated to be valid by a function's return value. No function in this volume of IEEE Std 1003.1-2001 shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by function calls or assignments to *errno* by other threads.

Some functions return an error number directly as the function value. These functions return a value of zero to indicate success.

If more than one error occurs in processing a function call, any one of the possible errors may be returned, as the order of detection is undefined.

Implementations may support additional errors not included in this list, may generate errors included in this list under circumstances other than those described here, or may contain extensions or limitations that prevent some errors from occurring. The ERRORS section on each reference page specifies whether an error shall be returned, or whether it may be returned. Implementations shall not generate a different error number from the ones described here for error conditions described in this volume of IEEE Std 1003.1-2001, but may generate additional errors unless explicitly disallowed for a particular function.

Each implementation shall document, in the conformance document, situations in which each of the optional conditions defined in IEEE Std 1003.1-2001 is detected. The conformance document may also contain statements that one or more of the optional error conditions are not detected.

For functions under the Threads option for which [EINTR] is not listed as a possible error condition in this volume of IEEE Std 1003.1-2001, an implementation shall not return an error code of [EINTR].

The following symbolic names identify the possible error numbers, in the context of the functions specifically defined in this volume of IEEE Std 1003.1-2001; these general descriptions are more precisely defined in the ERRORS sections of the functions that return them. Only these symbolic names should be used in programs, since the actual value of the error number is

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

unspecified. All values listed in this section shall be unique integer constant expressions with type **int** suitable for use in **#if** preprocessing directives, except as noted below. The values for all these names shall be found in the **<errno.h>** header defined in the Base Definitions volume of IEEE Std 1003.1-2001. The actual values are unspecified by this volume of IEEE Std 1003.1-2001.

[E2BIG]
Argument list too long. The sum of the number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of {ARG_MAX} bytes.

or:

Lack of space in an output buffer.

or:

Argument is greater than the system-imposed maximum.

[EACCES]
Permission denied. An attempt was made to access a file in a way forbidden by its file access permissions.

[EADDRINUSE]
Address in use. The specified address is in use.

[EADDRNOTAVAIL]
Address not available. The specified address is not available from the local system.

[EAFNOSUPPORT]
Address family not supported. The implementation does not support the specified address family, or the specified address is not a valid address for the address family of the specified socket.

[EAGAIN]
Resource temporarily unavailable. This is a temporary condition and later calls to the same routine may complete normally.

[EALREADY]
Connection already in progress. A connection request is already in progress for the specified socket.

[EBADF]
Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a read (write) request is made to a file that is only open for writing (reading).

[EBADMSG]
XSR Bad message. During a *read*(), *getmsg*(), *getpmsg*(), or *ioctl*() I_RECVFD request to a STREAMS device, a message arrived at the head of the STREAM that is inappropriate for the function receiving the message.

read() Message waiting to be read on a STREAM is not a data message.

getmsg() or *getpmsg*()
A file descriptor was received instead of a control message.

ioctl() Control or data information was received instead of a file descriptor when I_RECVFD was specified.

or:

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Bad Message. The implementation has detected a corrupted message.

[EBUSY]
Resource busy. An attempt was made to make use of a system resource that is not currently available, as it is being used by another process in a manner that would have conflicted with the request being made by this process.

[ECANCELED]
Operation canceled. The associated asynchronous operation was canceled before completion.

[ECHILD]
No child process. A *wait*( ) or *waitpid*( ) function was executed by a process that had no existing or unwaited-for child process.

[ECONNABORTED]
Connection aborted. The connection has been aborted.

[ECONNREFUSED]
Connection refused. An attempt to connect to a socket was refused because there was no process listening or because the queue of connection requests was full and the underlying protocol does not support retransmissions.

[ECONNRESET]
Connection reset. The connection was forcibly closed by the peer.

[EDEADLK]
Resource deadlock would occur. An attempt was made to lock a system resource that would have resulted in a deadlock situation.

[EDESTADDRREQ]
Destination address required. No bind address was established.

[EDOM]
Domain error. An input argument is outside the defined domain of the mathematical function (defined in the ISO C standard).

[EDQUOT]
Reserved.

[EEXIST]
File exists. An existing file was mentioned in an inappropriate context; for example, as a new link name in the *link*( ) function.

[EFAULT]
Bad address. The system detected an invalid address in attempting to use an argument of a call. The reliable detection of this error cannot be guaranteed, and when not detected may result in the generation of a signal, indicating an address violation, which is sent to the process.

[EFBIG]
File too large. The size of a file would exceed the maximum file size of an implementation or offset maximum established in the corresponding file description.

[EHOSTUNREACH]
Host is unreachable. The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

[EIDRM]
Identifier removed. Returned during XSI interprocess communication if an identifier has

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

been removed from the system.

[EILSEQ]
Illegal byte sequence. A wide-character code has been detected that does not correspond to a valid character, or a byte sequence does not form a valid wide-character code (defined in the ISO C standard).

[EINPROGRESS]
Operation in progress. This code is used to indicate that an asynchronous operation has not yet completed.

or:

O_NONBLOCK is set for the socket file descriptor and the connection cannot be immediately established.

[EINTR]
Interrupted function call. An asynchronous signal was caught by the process during the execution of an interruptible function. If the signal handler performs a normal return, the interrupted function call may return this condition (see the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**).

[EINVAL]
Invalid argument. Some invalid argument was supplied; for example, specifying an undefined signal in a *signal*( ) function or a *kill*( ) function.

[EIO]
Input/output error. Some physical input or output error has occurred. This error may be reported on a subsequent operation on the same file descriptor. Any other error-causing operation on the same file descriptor may cause the [EIO] error indication to be lost.

[EISCONN]
Socket is connected. The specified socket is already connected.

[EISDIR]
Is a directory. An attempt was made to open a directory with write mode specified.

[ELOOP]
Symbolic link loop. A loop exists in symbolic links encountered during pathname resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links are encountered during pathname resolution.

[EMFILE]
Too many open files. An attempt was made to open more than the maximum number of file descriptors allowed in this process.

[EMLINK]
Too many links. An attempt was made to have the link count of a single file exceed {LINK_MAX}.

[EMSGSIZE]
Message too large. A message sent on a transport provider was larger than an internal message buffer or some other network limit.

or:

Inappropriate message buffer length.

[EMULTIHOP]
Reserved.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[ENAMETOOLONG]
Filename too long. The length of a pathname exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX}. This error may also occur when pathname substitution, as a result of encountering a symbolic link during pathname resolution, results in a pathname string the size of which exceeds {PATH_MAX}.

[ENETDOWN]
Network is down. The local network interface used to reach the destination is down.

[ENETRESET]
The connection was aborted by the network.

[ENETUNREACH]
Network unreachable. No route to the network is present.

[ENFILE]
Too many files open in system. Too many files are currently open in the system. The system has reached its predefined limit for simultaneously open files and temporarily cannot accept requests to open another one.

[ENOBUFS]
No buffer space available. Insufficient buffer resources were available in the system to perform the socket operation.

XSR [ENODATA]
No message available. No message is available on the STREAM head read queue.

[ENODEV]
No such device. An attempt was made to apply an inappropriate function to a device; for example, trying to read a write-only device such as a printer.

[ENOENT]
No such file or directory. A component of a specified pathname does not exist, or the pathname is an empty string.

[ENOEXEC]
Executable file format error. A request is made to execute a file that, although it has the appropriate permissions, is not in the format required by the implementation for executable files.

[ENOLCK]
No locks available. A system-imposed limit on the number of simultaneous file and record locks has been reached and no more are currently available.

[ENOLINK]
Reserved.

[ENOMEM]
Not enough space. The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.

[ENOMSG]
No message of the desired type. The message queue does not contain a message of the required type during XSI interprocess communication.

[ENOPROTOOPT]
Protocol not available. The protocol option specified to *setsockopt*( ) is not supported by the implementation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[ENOSPC]
No space left on a device. During the *write*( ) function on a regular file or when extending a directory, there is no free space left on the device.

XSR [ENOSR]
No STREAM resources. Insufficient STREAMS memory resources are available to perform a STREAMS-related function. This is a temporary condition; it may be recovered from if other processes release resources.

XSR [ENOSTR]
Not a STREAM. A STREAM function was attempted on a file descriptor that was not associated with a STREAMS device.

[ENOSYS]
Function not implemented. An attempt was made to use a function that is not available in this implementation.

[ENOTCONN]
Socket not connected. The socket is not connected.

[ENOTDIR]
Not a directory. A component of the specified pathname exists, but it is not a directory, when a directory was expected.

[ENOTEMPTY]
Directory not empty. A directory other than an empty directory was supplied when an empty directory was expected.

[ENOTSOCK]
Not a socket. The file descriptor does not refer to a socket.

[ENOTSUP]
Not supported. The implementation does not support this feature of the Realtime Option Group.

[ENOTTY]
Inappropriate I/O control operation. A control function has been attempted for a file or special file for which the operation is inappropriate.

[ENXIO]
No such device or address. Input or output on a special file refers to a device that does not exist, or makes a request beyond the capabilities of the device. It may also occur when, for example, a tape drive is not on-line.

[EOPNOTSUPP]
Operation not supported on socket. The type of socket (address family or protocol) does not support the requested operation.

[EOVERFLOW]
Value too large to be stored in data type. An operation was attempted which would generate a value that is outside the range of values that can be represented in the relevant data type or that are allowed for a given data item.

[EPERM]
Operation not permitted. An attempt was made to perform an operation limited to processes with appropriate privileges or to the owner of a file or other resource.

[EPIPE]
Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

to read the data.

[EPROTO]
Protocol error. Some protocol error occurred. This error is device-specific, but is generally not related to a hardware failure.

[EPROTONOSUPPORT]
Protocol not supported. The protocol is not supported by the address family, or the protocol is not supported by the implementation.

[EPROTOTYPE]
Protocol wrong type for socket. The socket type is not supported by the protocol.

[ERANGE]
Result too large or too small. The result of the function is too large (overflow) or too small (underflow) to be represented in the available space (defined in the ISO C standard).

[EROFS]
Read-only file system. An attempt was made to modify a file or directory on a file system that is read-only.

[ESPIPE]
Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.

[ESRCH]
No such process. No process can be found corresponding to that specified by the given process ID.

[ESTALE]
Reserved.

XSR [ETIME]
STREAM *ioctl*( ) timeout. The timer set for a STREAMS *ioctl*( ) call has expired. The cause of this error is device-specific and could indicate either a hardware or software failure, or a timeout value that is too short for the specific operation. The status of the *ioctl*( ) operation is unspecified.

[ETIMEDOUT]
Connection timed out. The connection to a remote machine has timed out. If the connection timed out during execution of the function that reported this error (as opposed to timing out prior to the function being called), it is unspecified whether the function has completed some or all of the documented behavior associated with a successful completion of the function.

or:

Operation timed out. The time limit associated with the operation was exceeded before the operation completed.

[ETXTBSY]
Text file busy. An attempt was made to execute a pure-procedure program that is currently open for writing, or an attempt has been made to open for writing a pure-procedure program that is being executed.

[EWOULDBLOCK]
Operation would block. An operation on a socket marked as non-blocking has encountered a situation such as no data available that otherwise would have caused the function to suspend execution.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A conforming implementation may assign the same values for [EWOULDBLOCK] and [EAGAIN].

[EXDEV]
Improper link. A link to a file on another file system was attempted.

2.3.1 Additional Error Numbers

Additional implementation-defined error numbers may be defined in **<errno.h>**.

2.4 Signal Concepts

2.4.1 Signal Generation and Delivery

A signal is said to be ''generated'' for (or sent to) a process or thread when the event that causes the signal first occurs. Examples of such events include detection of hardware faults, timer expiration, RTS signals generated via the **sigevent** structure RTS and terminal activity, as well as invocations of the *kill*( ) and *sigqueue*( ) functions. In some circumstances, the same event generates signals for multiple processes.

At the time of generation, a determination shall be made whether the signal has been generated for the process or for a specific thread within the process. Signals which are generated by some action attributable to a particular thread, such as a hardware fault, shall be generated for the thread that caused the signal to be generated. Signals that are generated in association with a process ID or process group ID or an asynchronous event, such as terminal activity, shall be generated for the process.

Each process has an action to be taken in response to each signal defined by the system (see Section 2.4.3 (on page 30)). A signal is said to be ''delivered'' to a process when the appropriate action for the process and signal is taken. A signal is said to be ''accepted'' by a process when the signal is selected and returned by one of the *sigwait*( ) functions.

During the time between the generation of a signal and its delivery or acceptance, the signal is said to be ''pending''. Ordinarily, this interval cannot be detected by an application. However, a signal can be ''blocked'' from delivery to a thread. If the action associated with a blocked signal is anything other than to ignore the signal, and if that signal is generated for the thread, the signal shall remain pending until it is unblocked, it is accepted when it is selected and returned by a call to the *sigwait*( ) function, or the action associated with it is set to ignore the signal. Signals generated for the process shall be delivered to exactly one of those threads within the process which is in a call to a *sigwait*( ) function selecting that signal or has not blocked delivery of the signal. If there are no threads in a call to a *sigwait*( ) function selecting that signal, and if all threads within the process block delivery of the signal, the signal shall remain pending on the process until a thread calls a *sigwait*( ) function selecting that signal, a thread unblocks delivery of the signal, or the action associated with the signal is set to ignore the signal. If the action associated with a blocked signal is to ignore the signal and if that signal is generated for the process, it is unspecified whether the signal is discarded immediately upon generation or remains pending.

Each thread has a ''signal mask'' that defines the set of signals currently blocked from delivery to it. The signal mask for a thread shall be initialized from that of its parent or creating thread, or from the corresponding thread in the parent process if the thread was created as the result of a call to *fork*( ). The *pthread_sigmask*( ), *sigaction*( ), *sigprocmask*( ), and *sigsuspend*( ) functions control the manipulation of the signal mask.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The determination of which action is taken in response to a signal is made at the time the signal is delivered, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated. If a subsequent occurrence of a pending signal is generated, it is implementation-defined as to whether the signal is delivered or accepted more than once RTS in circumstances other than those in which queuing is required under the Realtime Signals Extension option. The order in which multiple, simultaneously pending signals outside the range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is generated for a process, all pending stop signals for that process shall be discarded. When SIGCONT is generated for a process that is stopped, the process shall be continued, even if the SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain pending until it is either unblocked or a stop signal is generated for the process.

An implementation shall document any condition not specified by this volume of IEEE Std 1003.1-2001 under which the implementation generates signals.

2.4.2 Realtime Signal Generation and Delivery

RTS This section describes extensions to support realtime signal generation and delivery. This functionality is dependent on support of the Realtime Signals Extension option (and the rest of this section is not further shaded for this option).

Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O completion, interprocess message arrival, and the *sigqueue*( ) function, support the specification of an application-defined value, either explicitly as a parameter to the function or in a **sigevent** structure parameter. The **sigevent** structure is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	*sigev_notify*	Notification type.
int	*sigev_signo*	Signal number.
union sigval	*sigev_value*	Signal value.
void(*)(unsigned sigval)	*sigev_notify_function*	Notification function.
(pthread_attr_t*)	*sigev_notify_attributes*	Notification attributes.

The *sigev_notify* member specifies the notification mechanism to use when an asynchronous event occurs. This volume of IEEE Std 1003.1-2001 defines the following values for the *sigev_notify* member:

SIGEV_NONE No asynchronous notification shall be delivered when the event of interest occurs.

SIGEV_SIGNAL The signal specified in *sigev_signo* shall be generated for the process when the event of interest occurs. If the implementation supports the Realtime Signals Extension option and if the SA_SIGINFO flag is set for that signal number, then the signal shall be queued to the process and the value specified in *sigev_value* shall be the *si_value* component of the generated signal. If SA_SIGINFO is not set for that signal number, it is unspecified whether the signal is queued and what value, if any, is sent.

SIGEV_THREAD A notification function shall be called to perform notification.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

An implementation may define additional notification mechanisms.

The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the application-defined value to be passed to the signal-catching function at the time of the signal delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

The **sigval** union is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	*sival_int*	Integer signal value.
void*	*sival_ptr*	Pointer signal value.

The *sival_int* member shall be used when the application-defined value is of type **int**; the *sival_ptr* member shall be used when the application-defined value is a pointer.

When a signal is generated by the *sigqueue*( ) function or any signal-generating function that supports the specification of an application-defined value, the signal shall be marked pending and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along with the application-specified signal value. Multiple occurrences of signals so generated are queued in FIFO order. It is unspecified whether signals so generated are queued when the SA_SIGINFO flag is not set for that signal.

Signals generated by the *kill*( ) function or other events that cause signals to occur, such as detection of hardware faults, *alarm*( ) timer expiration, or terminal activity, and for which the implementation does not support queuing, shall have no effect on signals already queued for the same signal number.

When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the behavior shall be as if the implementation delivers the pending unblocked signal with the lowest signal number within that range. No other ordering of signal delivery is specified.

If, when a pending signal is delivered, there are additional signals queued to that signal number, the signal shall remain pending. Otherwise, the pending indication shall be reset.

Multi-threaded programs can use an alternate event notification mechanism. When a notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

The function shall be executed in an environment as if it were the *start_routine* for a newly created thread with thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes* is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this thread is implementation-defined.

2.4.3 Signal Actions

There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of the *main*( ) routine (see the *exec* functions). The actions prescribed by these values are as follows:

SIG_DFL Signal-specific default action.

RTS The default actions for the signals defined in this volume of IEEE Std 1003.1-2001 are specified under **<signal.h>**. If the Realtime Signals Extension option is supported, the default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX shall be to terminate the process abnormally.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag. While a process is stopped, any additional signals that are sent to the process shall not be delivered until the process is continued, except SIGKILL which always terminates the receiving process. A process that is a member of an orphaned process group shall not be allowed to stop in response to the SIGTSTP, SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a process, the signal shall be discarded.

Setting a signal action to SIG_DFL for a signal that is pending, and whose default action is to ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded, whether or not it is blocked. If the Realtime Signals Extension option is supported, any queued values pending shall be discarded and the resources used to queue them shall be released and returned to the system for other use.

The default action for SIGCONT is to resume execution at the point where the process was stopped, after first handling any pending unblocked signals.

When a stopped process is continued, a SIGCHLD signal may be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag.

SIG_IGN Ignore signal.

Delivery of the signal shall have no effect on the process. The behavior of a process is undefined after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by *kill*( ), *sigqueue*( ), or *raise*( ).

The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG_IGN.

Setting a signal action to SIG_IGN for a signal that is pending shall cause the pending signal to be discarded, whether or not it is blocked.

If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified, except as specified below.

If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and *wait*( ), *waitid*( ), and *waitpid*( ) shall fail and set *errno* to [ECHILD].

If the Realtime Signals Extension option is supported, any queued values pending shall be discarded and the resources used to queue them shall be released and made available to queue other signals.

pointer to a function

Catch signal.

On delivery of the signal, the receiving process is to execute the signal-catching function at the specified address. After returning from the signal-catching function, the receiving process shall resume execution at the point at which it was interrupted.

If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as a C-language function call as follows:

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
void func(int signo);
```

XSI | RTS If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be entered as a C-language function call as follows:

```
void func(int signo, siginfo_t *info, void *context);
```

where *func* is the specified signal-catching function, *signo* is the signal number of the signal being delivered, and *info* is a pointer to a **siginfo_t** structure defined in **<signal.h>** containing at least the following members:

Member Type	Member Name	Description
int	*si_signo*	Signal number.
int	*si_code*	Cause of the signal.
union sigval	*si_value*	Signal value.

The *si_signo* member shall contain the signal number. This shall be the same as the *signo* parameter. The *si_code* member shall contain a code identifying the cause of the signal. The following values are defined for *si_code*:

SI_USER The signal was sent by the *kill*( ) function. The implementation may set *si_code* to SI_USER if the signal was sent by the *raise*( ) or *abort*( ) functions or any similar functions provided as implementation extensions.

RTS SI_QUEUE The signal was sent by the *sigqueue*( ) function.

RTS SI_TIMER The signal was generated by the expiration of a timer set by *timer_settime*( ).

RTS SI_ASYNCIO The signal was generated by the completion of an asynchronous I/O request.

RTS SI_MESGQ The signal was generated by the arrival of a message on an empty message queue.

If the signal was not generated by one of the functions or events listed above, the *si_code* shall be set to an implementation-defined value that is not equal to any of the values defined above.

RTS If the Realtime Signals Extension is supported, and *si_code* is one of SI_QUEUE, SI_TIMER, SI_ASYNCIO, or SI_MESGQ, then *si_value* shall contain the application-specified signal value. Otherwise, the contents of *si_value* are undefined.

The behavior of a process is undefined after it returns normally from a signal-
XSI catching function for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not
RTS generated by *kill*( ), *sigqueue*( ), or *raise*( ).

The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

If a process establishes a signal-catching function for the SIGCHLD signal while it has a terminated child process for which it has not waited, it is unspecified whether a SIGCHLD signal is generated to indicate that child process.

When signal-catching functions are invoked asynchronously with process execution, the behavior of some of the functions defined by this volume of IEEE Std 1003.1-2001 is unspecified if they are called from a signal-catching function.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

The following table defines a set of functions that shall be either reentrant or non-interruptible by signals and shall be async-signal-safe. Therefore applications may invoke them, without restriction, from signal-catching functions:

_Exit()	*fpathconf*()	*read*()	*sigset*()
_exit()	*fstat*()	*readlink*()	*sigsuspend*()
abort()	*fsync*()	*recv*()	*socket*()
accept()	*ftruncate*()	*recvfrom*()	*socketpair*()
access()	*getegid*()	*recvmsg*()	*stat*()
aio_error()	*geteuid*()	*rename*()	*symlink*()
aio_return()	*getgid*()	*rmdir*()	*sysconf*()
aio_suspend()	*getgroups*()	*select*()	*tcdrain*()
alarm()	*getpeername*()	*sem_post*()	*tcflow*()
bind()	*getpgrp*()	*send*()	*tcflush*()
cfgetispeed()	*getpid*()	*sendmsg*()	*tcgetattr*()
cfgetospeed()	*getppid*()	*sendto*()	*tcgetpgrp*()
cfsetispeed()	*getsockname*()	*setgid*()	*tcsendbreak*()
cfsetospeed()	*getsockopt*()	*setpgid*()	*tcsetattr*()
chdir()	*getuid*()	*setsid*()	*tcsetpgrp*()
chmod()	*kill*()	*setsockopt*()	*time*()
chown()	*link*()	*setuid*()	*timer_getoverrun*()
clock_gettime()	*listen*()	*shutdown*()	*timer_gettime*()
close()	*lseek*()	*sigaction*()	*timer_settime*()
connect()	*lstat*()	*sigaddset*()	*times*()
creat()	*mkdir*()	*sigdelset*()	*umask*()
dup()	*mkfifo*()	*sigemptyset*()	*uname*()
dup2()	*open*()	*sigfillset*()	*unlink*()
execle()	*pathconf*()	*sigismember*()	*utime*()
execve()	*pause*()	*sleep*()	*wait*()
fchmod()	*pipe*()	*signal*()	*waitpid*()
fchown()	*poll*()	*sigpause*()	*write*()
fcntl()	*posix_trace_event*()	*sigpending*()	
fdatasync()	*pselect*()	*sigprocmask*()	
fork()	*raise*()	*sigqueue*()	

All functions not in the above table are considered to be unsafe with respect to signals. In the presence of signals, all functions defined by this volume of IEEE Std 1003.1-2001 shall behave as defined when called from or interrupted by a signal-catching function, with a single exception: when a signal interrupts an unsafe function and the signal-catching function calls an unsafe function, the behavior is undefined.

When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or continue, the entire process shall be terminated, stopped, or continued, respectively.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.4.4 Signal Effects on Other Functions

Signals affect the behavior of certain functions defined by this volume of IEEE Std 1003.1-2001 if delivered to a process while it is executing such a function. If the action of the signal is to terminate the process, the process shall be terminated and the function shall not return. If the action of the signal is to stop the process, the process shall stop until continued or terminated. Generation of a SIGCONT signal for the process shall cause the process to be continued, and the original function shall continue at the point the process was stopped. If the action of the signal is to invoke a signal-catching function, the signal-catching function shall be invoked; in this case the original function is said to be ''interrupted'' by the signal. If the signal-catching function executes a **return** statement, the behavior of the interrupted function shall be as described individually for that function, except as noted for unsafe functions. Signals that are ignored shall not affect the behavior of any function; signals that are blocked shall not affect the behavior of any function until they are unblocked and then delivered, except as specified for the *sigpending*( ) and *sigwait*( ) functions.

2.5 Standard I/O Streams

A stream is associated with an external file (which may be a physical device) by ''opening'' a file, which may involve ''creating'' a new file. Creating an existing file causes its former contents to be discarded if necessary. If a file can support positioning requests (such as a disk file, as opposed to a terminal), then a ''file position indicator'' associated with the stream is positioned at the start (byte number 0) of the file, unless the file is opened with append mode, in which case it is implementation-defined whether the file position indicator is initially positioned at the beginning or end of the file. The file position indicator is maintained by subsequent reads, writes, and positioning requests, to facilitate an orderly progression through the file. All input takes place as if bytes were read by successive calls to *fgetc*( ); all output takes place as if bytes were written by successive calls to *fputc*( ).

When a stream is ''unbuffered'', bytes are intended to appear from the source or at the destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a block. When a stream is ''fully buffered'', bytes are intended to be transmitted as a block when a buffer is filled. When a stream is ''line buffered'', bytes are intended to be transmitted as a block when a newline byte is encountered. Furthermore, bytes are intended to be transmitted as a block when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line-buffered stream that requires the transmission of bytes. Support for these characteristics is implementation-defined, and may be affected via *setbuf*( ) and *setvbuf*( ).

A file may be disassociated from a controlling stream by ''closing'' the file. Output streams are flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed (including the standard streams).

A file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at its start). If the *main*( ) function returns to its original caller, or if the *exit*( ) function is called, all open files are closed (hence all output streams are flushed) before program termination. Other paths to program termination, such as calling *abort*( ), need not close all files properly.

The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE** object need not necessarily serve in place of the original.

At program start-up, three streams are predefined and need not be opened explicitly: *standard input* (for reading conventional input), *standard output* (for writing conventional output), and

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

standard error (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

2.5.1 Interaction of File Descriptors and Standard I/O Streams

CX This section describes the interaction of file descriptors and standard I/O streams. This functionality is an extension to the ISO C standard (and the rest of this section is not further CX shaded).

An open file description may be accessed through a file descriptor, which is created using functions such as *open*( ) or *pipe*( ), or through a stream, which is created using functions such as *fopen*( ) or *popen*( ). Either a file descriptor or a stream is called a ''handle'' on the open file description to which it refers; an open file description may have several handles.

Handles can be created or destroyed by explicit user action, without affecting the underlying open file description. Some of the ways to create them include *fcntl*( ), *dup*( ), *fdopen*( ), *fileno*( ), and *fork*( ). They can be destroyed by at least *fclose*( ), *close*( ), and the *exec* functions.

A file descriptor that is never used in an operation that could affect the file offset (for example, *read*( ), *write*( ), or *lseek*( )) is not considered a handle for this discussion, but could give rise to one (for example, as a consequence of *fdopen*( ), *dup*( ), or *fork*( )). This exception does not include the file descriptor underlying a stream, whether created with *fopen*( ) or *fdopen*( ), so long as it is not used directly by the application to affect the file offset. The *read*( ) and *write*( ) functions implicitly affect the file offset; *lseek*( ) explicitly affects it.

The result of function calls involving any one handle (the ''active handle'') is defined elsewhere in this volume of IEEE Std 1003.1-2001, but if two or more handles are used, and any one of them is a stream, the application shall ensure that their actions are coordinated as described below. If this is not done, the result is undefined.

A handle which is a stream is considered to be closed when either an *fclose*( ) or *freopen*( ) is executed on it (the result of *freopen*( ) is a new stream, which cannot be a handle on the same open file description as its previous value), or when the process owning that stream terminates with *exit*( ), *abort*( ), or due to a signal. A file descriptor is closed by *close*( ), *_exit*( ), or the *exec* functions when FD_CLOEXEC is set on that file descriptor.

For a handle to become the active handle, the application shall ensure that the actions below are performed between the last use of the handle (the current active handle) and the first use of the second handle (the future active handle). The second handle then becomes the active handle. All activity by the application affecting the file offset on the first handle shall be suspended until it again becomes the active file handle. (If a stream function has as an underlying function one that affects the file offset, the stream function shall be considered to affect the file offset.)

The handles need not be in the same process for these rules to apply.

Note that after a *fork*( ), two handles exist where one existed before. The application shall ensure that, if both handles can ever be accessed, they are both in a state where the other could become the active handle first. The application shall prepare for a *fork*( ) exactly as if it were a change of active handle. (If the only action performed by one of the processes is one of the *exec* functions or *_exit*( ) (not *exit*( )), the handle is never accessed in that process.)

For the first handle, the first applicable condition below applies. After the actions required below are taken, if the handle is still open, the application can close it.

- If it is a file descriptor, no action is required.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- If the only further action to be performed on any handle to this open file descriptor is to close it, no action need be taken.
- If it is a stream which is unbuffered, no action need be taken.
- If it is a stream which is line buffered, and the last byte written to the stream was a <newline> (that is, as if a:

  ```
  putc('\n')
  ```

  was the most recent operation on that stream), no action need be taken.
- If it is a stream which is open for writing or appending (but not also open for reading), the application shall either perform an *fflush*( ), or the stream shall be closed.
- If the stream is open for reading and it is at the end of the file (*feof*( ) is true), no action need be taken.
- If the stream is open with a mode that allows reading and the underlying open file description refers to a device that is capable of seeking, the application shall either perform an *fflush*( ), or the stream shall be closed.

Otherwise, the result is undefined.

For the second handle:

- If any previous active handle has been used by a function that explicitly changed the file offset, except as required above for the first handle, the application shall perform an *lseek*( ) or *fseek*( ) (as appropriate to the type of handle) to an appropriate location.

If the active handle ceases to be accessible before the requirements on the first handle, above, have been met, the state of the open file description becomes undefined. This might occur during functions such as a *fork*( ) or *_exit*( ).

The *exec* functions make inaccessible all streams that are open at the time they are called, independent of which streams or file descriptors may be available to the new process image.

When these rules are followed, regardless of the sequence of handles used, implementations shall ensure that an application, even one consisting of several processes, shall yield correct results: no data shall be lost or duplicated when writing, and all data shall be written in order, except as requested by seeks. It is implementation-defined whether, and under what conditions, all input is seen exactly once.

If the rules above are not followed, the result is unspecified.

Each function that operates on a stream is said to have zero or more ‘‘underlying functions’’. This means that the stream function shares certain traits with the underlying functions, but does not require that there be any relation between the implementations of the stream function and its underlying functions.

2.5.2 Stream Orientation and Encoding Rules

For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an ‘‘orientation’’. After a stream is associated with an external file, but before any operations are performed on it, the stream is without orientation. Once a wide-character input/output function has been applied to a stream without orientation, the stream shall become ‘‘wide-oriented’’. Similarly, once a byte input/output function has been applied to a stream without orientation, the stream shall become ‘‘byte-oriented’’. Only a call to the *freopen*( ) function or the *fwide*( ) function can otherwise alter the orientation of a stream.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A successful call to *freopen*( ) shall remove any orientation. The three predefined streams *standard input*, *standard output*, and *standard error* shall be unoriented at program start-up.

Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character input/output functions cannot be applied to a byte-oriented stream. The remaining stream operations shall not affect and shall not be affected by a stream's orientation, except for the following additional restriction:

- For wide-oriented streams, after a successful call to a file-positioning function that leaves the file position indicator prior to the end-of-file, a wide-character output function can overwrite a partial character; any file contents beyond the byte(s) written are henceforth undefined.

Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state of the stream. A successful call to *fgetpos*( ) shall store a representation of the value of this **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos*( ) using the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as the position within the controlled stream.

Implementations that support multiple encoding rules associate an encoding rule with the stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the current locale at the time when the stream becomes wide-oriented. As with the stream's orientation, the encoding rule associated with a stream cannot be changed once it has been set, except by a successful call to *freopen*( ) which clears the encoding rule and resets the orientation to unoriented.

Although wide-oriented streams are conceptually sequences of wide characters, the external file associated with a wide-oriented stream is a sequence of (possibly multi-byte) characters generalized as follows:

- Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte encodings valid for use internal to the program).
- A file need not begin nor end in the initial shift state.

Moreover, the encodings used for characters may differ among files. Both the nature and choice of such encodings are implementation-defined.

The wide-character input functions read characters from the stream and convert them to wide characters as if they were read by successive calls to the *fgetwc*( ) function. Each conversion shall occur as if by a call to the *mbrtowc*( ) function, with the conversion state described by the stream's own **mbstate_t** object, except the encoding rule associated with the stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current locale. CX

The wide-character output functions convert wide characters to (possibly multi-byte) characters and write them to the stream as if they were written by successive calls to the *fputwc*( ) function. Each conversion shall occur as if by a call to the *wcrtomb*( ) function, with the conversion state described by the stream's own **mbstate_t** object, except the encoding rule associated with the stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current locale. CX

An ''encoding error'' shall occur if the character sequence presented to the underlying *mbrtowc*( ) function does not form a valid (generalized) character, or if the code value passed to the underlying *wcrtomb*( ) function does not correspond to a valid (generalized) character. The wide-character input/output functions and the byte input/output functions store the value of the macro [EILSEQ] in *errno* if and only if an encoding error occurs.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.6 STREAMS

XSR STREAMS functionality is provided on implementations supporting the XSI STREAMS Option Group. This functionality is dependent on support of the XSI STREAMS option (and the rest of this section is not further shaded for this option).

STREAMS provides a uniform mechanism for implementing networking services and other character-based I/O. The STREAMS function provides direct access to protocol modules. STREAMS modules are unspecified objects. Access to STREAMS modules is provided by interfaces in IEEE Std 1003.1-2001. Creation of STREAMS modules is outside the scope of IEEE Std 1003.1-2001.

A STREAM is typically a full-duplex connection between a process and an open device or pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex connection between two processes. The STREAM itself exists entirely within the implementation and provides a general character I/O function for processes. It optionally includes one or more intermediate processing modules that are interposed between the process end of the STREAM (STREAM head) and a device driver at the end of the STREAM (STREAM end).

STREAMS I/O is based on messages. There are three types of message:

- *Data messages* containing actual data for input or output
- *Control data* containing instructions for the STREAMS modules and underlying implementation
- Other messages, which include file descriptors

The interface between the STREAM and the rest of the implementation is provided by a set of functions at the STREAM head. When a process calls *write*(), *writev*(), *putmsg*(), *putpmsg*(), or *ioctl*(), messages are sent down the STREAM, and *read*(), *readv*(), *getmsg*(), or *getpmsg*() accepts data from the STREAM and passes it to a process. Data intended for the device at the downstream end of the STREAM is packaged into messages and sent downstream, while data and signals from the device are composed into messages by the device driver and sent upstream to the STREAM head.

When a STREAMS-based device is opened, a STREAM shall be created that contains the STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an implementation, when a pipe is created, two STREAMS shall be created, each containing a STREAM head. Other modules are added to the STREAM using *ioctl*(). New modules are ''pushed'' onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the STREAM was a push-down stack.

Priority

Message types are classified according to their queuing priority and may be *normal* (non-priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that determines its ordering when placed on a queue. Normal messages have a priority band of 0 and shall always be placed at the end of the queue following all other messages in the queue. High-priority messages are always placed at the head of a queue, but shall be discarded if there is already a high-priority message in the queue. Their priority band shall be ignored; they are high-priority by virtue of their type. Priority messages have a priority band greater than 0. Priority messages are always placed after any messages of the same or higher priority. High-priority and priority messages are used to send control and data information outside the normal flow of control. By convention, high-priority messages shall not be affected by flow control. Normal and priority messages have separate flow controls.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Message Parts

A process may access STREAMS messages that contain a data part, control part, or both. The data part is that information which is transmitted over the communication medium and the control information is used by the local STREAMS modules. The other types of messages are used between modules and are not accessible to processes. Messages containing only a data part are accessible via *putmsg*( ), *putpmsg*( ), *getmsg*( ), *getpmsg*( ), *read*( ), *readv*( ), *write*( ), or *writev*( ). Messages containing a control part with or without a data part are accessible via calls to *putmsg*( ), *putpmsg*( ), *getmsg*( ), or *getpmsg*( ).

2.6.1 Accessing STREAMS

A process accesses STREAMS-based files using the standard functions *close*( ), *ioctl*( ), *getmsg*( ), *getpmsg*( ), *open*( ), *pipe*( ), *poll*( ), *putmsg*( ), *putpmsg*( ), *read*( ), or *write*( ). Refer to the applicable function definitions for general properties and errors.

Calls to *ioctl*( ) shall perform control functions on the STREAM associated with the file descriptor *fildes*. The control functions may be performed by the STREAM head, a STREAMS module, or the STREAMS driver for the STREAM.

STREAMS modules and drivers can detect errors, sending an error message to the STREAM head, thus causing subsequent functions to fail and set *errno* to the value specified in the message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl*( ) request alone by sending a negative acknowledgement message to the STREAM head. This shall cause just the pending *ioctl*( ) request to fail and set *errno* to the value specified in the message.

2.7 XSI Interprocess Communication

XSI This section describes extensions to support interprocess communication. This functionality is dependent on support of the XSI extension (and the rest of this section is not further shaded for this option).

The following message passing, semaphore, and shared memory services form an XSI interprocess communication facility. Certain aspects of their operation are common, and are defined as follows.

IPC Functions		
msgctl( )	*semctl*( )	*shmctl*( )
msgget( )	*semget*( )	*shmdt*( )
msgrcv( )	*semop*( )	*shmget*( )
msgsnd( )	*shmat*( )	

Another interprocess communication facility is provided by functions in the Realtime Option Group; see Section 2.8 (on page 41).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.7.1 IPC General Description

Each individual shared memory segment, message queue, and semaphore set shall be identified by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a semaphore identifier, *semid*, and a message queue identifier, *msqid*. The identifiers shall be returned by calls to *shmget*( ), *semget*( ), and *msgget*( ), respectively.

Associated with each identifier is a data structure which contains data related to the operations which may be or may have been performed; see the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/shm.h>**, **<sys/sem.h>**, and **<sys/msg.h>** for their descriptions.

Each of the data structures contains both ownership information and an **ipc_perm** structure (see the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/ipc.h>**) which are used in conjunction to determine whether or not read/write (read/alter for semaphores) permissions should be granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as a bit field which determines the permissions.

The values of the bits are given below in octal notation.

Bit	Meaning
0400	Read by user.
0200	Write by user.
0040	Read by group.
0020	Write by group.
0004	Read by others.
0002	Write by others.

The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which service is being used. In each case, read and write/alter permissions shall be granted to a process if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as appropriate):

- The process has appropriate privileges.
- The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data structure associated with the IPC identifier, and the appropriate bit of the *user* field in *xxx_perm.mode* is set.
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* but the effective group ID of the process matches *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, and the appropriate bit of the *group* field in *xxx_perm.mode* is set.
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* and the effective group ID of the process does not match *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, but the appropriate bit of the *other* field in *xxx_perm.mode* is set.

Otherwise, the permission shall be denied.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.8 Realtime

This section defines functions to support the source portability of applications with realtime requirements. The presence of many of these functions is dependent on support for implementation options described in the text.

The specific functional areas included in this section and their scope include the following. Full definitions of these terms can be found in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 3, Definitions.

- Semaphores
- Process Memory Locking
- Memory Mapped Files and Shared Memory Objects
- Priority Scheduling
- Realtime Signal Extension
- Timers
- Interprocess Communication
- Synchronized Input and Output
- Asynchronous Input and Output

All the realtime functions defined in this volume of IEEE Std 1003.1-2001 are portable, although some of the numeric parameters used by an implementation may have hardware dependencies.

2.8.1 Realtime Signals

RTS Realtime signal generation and delivery is dependent on support for the Realtime Signals Extension option.

See Section 2.4.2 (on page 29).

2.8.2 Asynchronous I/O

AIO The functionality described in this section is dependent on support of the Asynchronous Input and Output option (and the rest of this section is not further shaded for this option).

An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O functions. It is defined in the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>** and has at least the following members:

Member Type	Member Name	Description
int	*aio_fildes*	File descriptor.
off_t	*aio_offset*	File offset.
volatile void*	*aio_buf*	Location of buffer.
size_t	*aio_nbytes*	Length of transfer.
int	*aio_reqprio*	Request priority ofset.
struct sigevent	*aio_sigevent*	Signal number and value.
int	*aio_lio_opcode*	Operation to be performed.

The *aio_fildes* element is the file descriptor on which the asynchronous operation is performed.

If O_APPEND is not set for the file descriptor *aio_fildes* and if *aio_fildes* is associated with a device that is capable of seeking, then the requested operation takes place at the absolute position in the file as given by *aio_offset*, as if *lseek*( ) were called immediately prior to the

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to SEEK_SET. If O_APPEND is set for the file descriptor, or if *aio_fildes* is associated with a device that is incapable of seeking, write operations append to the file in the same order as the calls were made, with the following exception: under implementation-defined circumstances, such as operation on a multi-processor or when requests of differing priorities are submitted at the same time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming application to determine whether this relaxation applies, all strictly conforming applications which rely on ordering of output shall be written in such a way that they will operate correctly if the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf* elements are the same as the *nbyte* and *buf* arguments defined by *read*( ) and *write*( ), respectively.

If _POSIX_PRIORITIZED_IO and _POSIX_PRIORITY_SCHEDULING are defined, then asynchronous I/O is queued in priority order, with the priority of each asynchronous operation based on the current scheduling priority of the calling process. The *aio_reqprio* member can be used to lower (but not raise) the asynchronous I/O operation priority and is within the range zero through {AIO_PRIO_DELTA_MAX}, inclusive. Unless both _POSIX_PRIORITIZED_IO and _POSIX_PRIORITY_SCHEDULING are defined, the order of processing asynchronous I/O requests is unspecified. When both _POSIX_PRIORITIZED_IO and _POSIX_PRIORITY_SCHEDULING are defined, the order of processing of requests submitted by processes whose schedulers are not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is unspecified. The priority of an asynchronous request is computed as (process scheduling priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an indication of the desired order of execution of the request relative to other asynchronous I/O requests for this file. If _POSIX_PRIORITIZED_IO is defined, requests issued with the same priority to a character special file are processed by the underlying device in FIFO order; the order of processing of requests of the same priority issued to files that are not character special files is unspecified. Numerically higher priority values indicate requests of higher priority. The value of *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous I/O requests to the same file are blocked waiting for a resource required for that I/O operation, the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is implementation-defined. If _POSIX_PRIORITIZED_IO is defined, the implementation shall define for which files I/O prioritization is supported.

The *aio_sigevent* determines how the calling process shall be notified upon I/O completion, as specified in Section 2.4.1 (on page 28). If *aio_sigevent.sigev_notify* is SIGEV_NONE, then no signal shall be posted upon I/O completion, but the error status for the operation and the return status for the operation shall be set appropriately.

The *aio_lio_opcode* field is used only by the *lio_listio*( ) call. The *lio_listio*( ) call allows multiple asynchronous I/O operations to be submitted at a single time. The function takes as an argument an array of pointers to **aiocb** structures. Each **aiocb** structure indicates the operation to be performed (read or write) via the *aio_lio_opcode* field.

The address of the **aiocb** structure is used as a handle for retrieving the error status and return status of the asynchronous operation while it is in progress.

The **aiocb** structure and the data buffers associated with the asynchronous I/O operation are being used by the system for asynchronous I/O while, and only while, the error status of the asynchronous operation is equal to [EINPROGRESS]. Applications shall not modify the **aiocb** structure while the structure is being used by the system for asynchronous I/O.

The return status of the asynchronous operation is the number of bytes transferred by the I/O operation. If the error status is set to indicate an error completion, then the return status is set to

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the return value that the corresponding *read*( ), *write*( ), or *fsync*( ) call would have returned. When the error status is not equal to [EINPROGRESS], the return status shall reflect the return status of the corresponding synchronous operation.

2.8.3 Memory Management

2.8.3.1 Memory Locking

MLR Range memory locking operations are defined in terms of pages. Implementations may restrict the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

ML | MLR Memory locking guarantees the residence of portions of the address space. It is implementation-defined whether locking memory guarantees fixed translation between virtual addresses (as seen by the process) and physical addresses. Per-process memory locks are not inherited across a *fork*( ), and all memory locks owned by a process are unlocked upon *exec* or process termination. Unmapping of an address range removes any memory locks established on that address range by this process.

2.8.3.2 Memory Mapped Files

MF The functionality described in this section is dependent on support of the Memory Mapped Files option (and the rest of this section is not further shaded for this option).

Range memory mapping operations are defined in terms of pages. Implementations may restrict the size and alignment of range mappings to be on page-size boundaries. The page size, in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

Memory mapped files provide a mechanism that allows a process to access files by directly incorporating file data into its address space. Once a file is mapped into a process address space, the data can be manipulated as memory. If more than one process maps a file, its contents are shared among them. If the mappings allow shared write access, then data written into the memory object through the address space of one process appears in the address spaces of all processes that similarly map the same portion of the memory object.

SHM Shared memory objects are named regions of storage that may be independent of the file system and can be mapped into the address space of one or more processes to allow them to share the associated memory.

SHM An *unlink*( ) of a file or *shm_unlink*( ) of a shared memory object, while causing the removal of the name, does not unmap any mappings established for the object. Once the name has been removed, the contents of the memory object are preserved as long as it is referenced. The memory object remains referenced as long as a process has the memory object open or has some area of the memory object mapped.

2.8.3.3 Memory Protection

MPR MF The functionality described in this section is dependent on support of the Memory Protection and Memory Mapped Files option (and the rest of this section is not further shaded for these options).

When an object is mapped, various application accesses to the mapped region may result in signals. In this context, SIGBUS is used to indicate an error using the mapped object, and SIGSEGV is used to indicate a protection violation or misuse of an address:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- A mapping may be restricted to disallow some types of access.
- Write attempts to memory that was mapped without write access, or any access to memory mapped PROT_NONE, shall result in a SIGSEGV signal.
- References to unmapped addresses shall result in a SIGSEGV signal.
- Reference to whole pages within the mapping, but beyond the current length of the object, shall result in a SIGBUS signal.
- The size of the object is unaffected by access beyond the end of the object (even if a SIGBUS is not generated).

2.8.3.4 *Typed Memory Objects*

TYM The functionality described in this section is dependent on support of the Typed Memory Objects option (and the rest of this section is not further shaded for this option).

Implementations may support the Typed Memory Objects option without supporting the Memory Mapped Files option or the Shared Memory Objects option. Typed memory objects are implementation-configurable named storage pools accessible from one or more processors in a system, each via one or more ports, such as backplane buses, LANs, I/O channels, and so on. Each valid combination of a storage pool and a port is identified through a name that is defined at system configuration time, in an implementation-defined manner; the name may be independent of the file system. Using this name, a typed memory object can be opened and mapped into process address space. For a given storage pool and port, it is necessary to support both dynamic allocation from the pool as well as mapping at an application-supplied offset within the pool; when dynamic allocation has been performed, subsequent deallocation must be supported. Lastly, accessing typed memory objects from different ports requires a method for obtaining the offset and length of contiguous storage of a region of typed memory (dynamically allocated or not); this allows typed memory to be shared among processes and/or processors while being accessed from the desired port.

2.8.4 Process Scheduling

PS The functionality described in this section is dependent on support of the Process Scheduling option (and the rest of this section is not further shaded for this option).

Scheduling Policies

The scheduling semantics described in this volume of IEEE Std 1003.1-2001 are defined in terms of a conceptual model that contains a set of thread lists. No implementation structures are necessarily implied by the use of this conceptual model. It is assumed that no time elapses during operations described using this model, and therefore no simultaneous operations are possible. This model discusses only processor scheduling for runnable threads, but it should be noted that greatly enhanced predictability of realtime applications results if the sequencing of other resources takes processor scheduling policy into account.

There is, conceptually, one thread list for each priority. A runnable thread will be on the thread list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose of a scheduling policy is to define the allowable operations on this set of lists (for example, moving threads between and within lists).

Each process shall be controlled by an associated scheduling policy and priority. These parameters may be specified by explicit application execution of the *sched_setscheduler*( ) or *sched_setparam*( ) functions.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Each thread shall be controlled by an associated scheduling policy and priority. These parameters may be specified by explicit application execution of the *pthread_setschedparam*( ) function.

Associated with each policy is a priority range. Each policy definition shall specify the minimum priority range for that policy. The priority ranges for each policy may but need not overlap the priority ranges of other policies.

A conforming implementation shall select the thread that is defined as being at the head of the highest priority non-empty thread list to become a running thread, regardless of its associated policy. This thread is then removed from its thread list.

Four scheduling policies are specifically required. Other implementation-defined scheduling policies may be defined. The following symbols are defined in the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**:

SCHED_FIFO First in, first out (FIFO) scheduling policy.

SCHED_RR Round robin scheduling policy.

SS SCHED_SPORADIC Sporadic server scheduling policy.

SCHED_OTHER Another scheduling policy.

The values of these symbols shall be distinct.

SCHED_FIFO

Conforming implementations shall include a scheduling policy called the FIFO scheduling policy.

Threads scheduled under this policy are chosen from a thread list that is ordered by the time its threads have been on the list without being executed; generally, the head of the list is the thread that has been on the list the longest time, and the tail is the thread that has been on the list the shortest time.

Under the SCHED_FIFO policy, the modification of the definitional thread lists is as follows:

1. When a running thread becomes a preempted thread, it becomes the head of the thread list for its priority.
2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list for its priority.
3. When a running thread calls the *sched_setscheduler*( ) function, the process specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
4. When a running thread calls the *sched_setparam*( ) function, the priority of the process specified in the function call is modified to the priority specified by the *param* argument.
5. When a running thread calls the *pthread_setschedparam*( ) function, the thread specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
6. When a running thread calls the *pthread_setschedprio*( ) function, the thread specified in the function call is modified to the priority specified by the *prio* argument.
7. If a thread whose policy or priority has been modified other than by *pthread_setschedprio*( ) is a running thread or is runnable, it then becomes the tail of the thread list for its new priority.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8. If a thread whose policy or priority has been modified by *pthread_setschedprio*( ) is a running thread or is runnable, the effect on its position in the thread list depends on the direction of the modification, as follows:
   a. If the priority is raised, the thread becomes the tail of the thread list.
   b. If the priority is unchanged, the thread does not change position in the thread list.
   c. If the priority is lowered, the thread becomes the head of the thread list.
9. When a running thread issues the *sched_yield*( ) function, the thread becomes the tail of the thread list for its priority.
10. At no other time is the position of a thread with this scheduling policy within the thread lists affected.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions when SCHED_FIFO is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 priorities for this policy.

SCHED_RR

Conforming implementations shall include a scheduling policy called the ‘‘round robin’’ scheduling policy. This policy shall be identical to the SCHED_FIFO policy with the additional condition that when the implementation detects that a running thread has been executing as a running thread for a time period of the length returned by the *sched_rr_get_interval*( ) function or longer, the thread shall become the tail of its thread list and the head of that thread list shall be removed and made a running thread.

The effect of this policy is to ensure that if there are multiple SCHED_RR threads at the same priority, one of them does not monopolize the processor. An application should not rely only on the use of SCHED_RR to ensure application progress among multiple threads if the application includes threads using the SCHED_FIFO policy at the same or higher priority levels or SCHED_RR threads at a higher priority level.

A thread under this policy that is preempted and subsequently resumes execution as a running thread completes the unexpired portion of its round robin interval time period.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions when SCHED_RR is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 priorities for this policy.

SCHED_SPORADIC

SS | TSP The functionality described in this section is dependent on support of the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not further shaded for these options).

If _POSIX_SPORADIC_SERVER or _POSIX_THREAD_SPORADIC_SERVER is defined, the implementation shall include a scheduling policy identified by the value SCHED_SPORADIC.

The sporadic server policy is based primarily on two parameters: the replenishment period and the available execution capacity. The replenishment period is given by the *sched_ss_repl_period* member of the **sched_param** structure. The available execution capacity is initialized to the value given by the *sched_ss_init_budget* member of the same parameter. The sporadic server policy is identical to the SCHED_FIFO policy with some additional conditions that cause the thread’s assigned priority to be switched between the values specified by the *sched_priority* and

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sched_ss_low_priority members of the **sched_param** structure.

The priority assigned to a thread using the sporadic server scheduling policy is determined in the following manner: if the available execution capacity is greater than zero and the number of pending replenishment operations is strictly less than *sched_ss_max_repl*, the thread is assigned the priority specified by *sched_priority*; otherwise, the assigned priority shall be *sched_ss_low_priority*. If the value of *sched_priority* is less than or equal to the value of *sched_ss_low_priority*, the results are undefined. When active, the thread shall belong to the thread list corresponding to its assigned priority level, according to the mentioned priority assignment. The modification of the available execution capacity and, consequently of the assigned priority, is done as follows:

1. When the thread at the head of the *sched_priority* list becomes a running thread, its execution time shall be limited to at most its available execution capacity, plus the resolution of the execution time clock used for this scheduling policy. This resolution shall be implementation-defined.

2. Each time the thread is inserted at the tail of the list associated with *sched_priority*—because as a blocked thread it became runnable with priority *sched_priority* or because a replenishment operation was performed—the time at which this operation is done is posted as the *activation_time*.

3. When the running thread with assigned priority equal to *sched_priority* becomes a preempted thread, it becomes the head of the thread list for its priority, and the execution time consumed is subtracted from the available execution capacity. If the available execution capacity would become negative by this operation, it shall be set to zero.

4. When the running thread with assigned priority equal to *sched_priority* becomes a blocked thread, the execution time consumed is subtracted from the available execution capacity, and a replenishment operation is scheduled, as described in 6 and 7. If the available execution capacity would become negative by this operation, it shall be set to zero.

5. When the running thread with assigned priority equal to *sched_priority* reaches the limit imposed on its execution time, it becomes the tail of the thread list for *sched_ss_low_priority*, the execution time consumed is subtracted from the available execution capacity (which becomes zero), and a replenishment operation is scheduled, as described in 6 and 7.

6. Each time a replenishment operation is scheduled, the amount of execution capacity to be replenished, *replenish_amount*, is set equal to the execution time consumed by the thread since the *activation_time*. The replenishment is scheduled to occur at *activation_time* plus *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the replenishment operation is carried out immediately. Several replenishment operations may be pending at the same time, each of which will be serviced at its respective scheduled time. With the above rules, the number of replenishment operations simultaneously pending for a given thread that is scheduled under the sporadic server policy shall not be greater than *sched_ss_max_repl*.

7. A replenishment operation consists of adding the corresponding *replenish_amount* to the available execution capacity at the scheduled time. If, as a consequence of this operation, the execution capacity would become larger than *sched_ss_initial_budget*, it shall be rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it becomes the tail of the thread list for *sched_priority*.

Execution time is defined in Section 2.2.2 (on page 14).

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For this policy, changing the value of a CPU-time clock via *clock_settime*( ) shall have no effect on its behavior.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_min*( ) and *sched_get_priority_max*( ) functions when SCHED_SPORADIC is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 distinct priorities for this policy.

SCHED_OTHER

Conforming implementations shall include one scheduling policy identified as SCHED_OTHER (which may execute identically with either the FIFO or round robin scheduling policy). The effect of scheduling threads with the SCHED_OTHER policy in a system in which other threads SS are executing under SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is implementation-defined.

This policy is defined to allow strictly conforming applications to be able to indicate in a portable manner that they no longer need a realtime scheduling policy.

For threads executing under this policy, the implementation shall use only priorities within the range returned by the *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions when SCHED_OTHER is provided as the parameter.

2.8.5 Clocks and Timers

TMR The functionality described in this section is dependent on support of the Timers option (and the rest of this section is not further shaded for this option).

The **<time.h>** header defines the types and manifest constants used by the timing facility.

Time Value Specification Structures

Many of the timing facility functions accept or return time value specifications. A time value structure **timespec** specifies a single time value and includes at least the following members:

Member Type	Member Name	Description
time_t	*tv_sec*	Seconds.
long	*tv_nsec*	Nanoseconds.

The *tv_nsec* member is only valid if greater than or equal to zero, and less than the number of nanoseconds in a second (1 000 million). The time interval described by this structure is (*tv_sec* * 10^9 + *tv_nsec*) nanoseconds.

A time value structure **itimerspec** specifies an initial timer value and a repetition interval for use by the per-process timer functions. This structure includes at least the following members:

Member Type	Member Name	Description
struct timespec	*it_interval*	Timer period.
struct timespec	*it_value*	Timer expiration.

If the value described by *it_value* is non-zero, it indicates the time to or time of the next timer expiration (for relative and absolute timer values, respectively). If the value described by *it_value* is zero, the timer shall be disarmed.

If the value described by *it_interval* is non-zero, it specifies an interval which shall be used in reloading the timer when it expires; that is, a periodic timer is specified. If the value described by

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

it_interval is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is specified.

Timer Event Notification Control Block

RTS Per-process timers may be created that notify the process of timer expirations by queuing a realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, is used in creating such a timer. The **sigevent** structure contains the signal number and an application-specific data value which shall be used when notifying the calling process of timer expiration events.

Manifest Constants

The following constants are defined in the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**:

CLOCK_REALTIME The identifier for the system-wide realtime clock.

TIMER_ABSTIME Flag indicating time is absolute with respect to the clock associated with a timer.

MON CLOCK_MONOTONIC The identifier for the system-wide monotonic clock, which is defined as a clock whose value cannot be set via *clock_settime*( ) and which cannot have backward clock jumps. The maximum possible clock jump is implementation-defined.

MON The maximum allowable resolution for CLOCK_REALTIME and CLOCK_MONOTONIC clocks and all time services based on these clocks is represented by {_POSIX_CLOCKRES_MIN} and shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of resolution for these clocks to provide finer granularity time bases. The actual resolution supported by an implementation for a specific clock is obtained using the *clock_getres*( ) function. If the actual resolution supported for a time service based on one of these clocks differs from the resolution supported for that clock, the implementation shall document this difference.

MON The minimum allowable maximum value for CLOCK_REALTIME and CLOCK_MONOTONIC clocks and all absolute time services based on them is the same as that defined by the ISO C standard for the **time_t** type. If the maximum value supported by a time service based on one of these clocks differs from the maximum value supported by that clock, the implementation shall document this difference.

Execution Time Monitoring

CPT If _POSIX_CPUTIME is defined, process CPU-time clocks shall be supported in addition to the clocks described in **Manifest Constants**.

TCT If _POSIX_THREAD_CPUTIME is defined, thread CPU-time clocks shall be supported.

CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.117, CPU Time (Execution Time). The mechanism used to measure execution time is described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.9, Measurement of Execution Time.

CPT If _POSIX_CPUTIME is defined, the following constant of the type **clockid_t** is defined in **<time.h>**:

CLOCK_PROCESS_CPUTIME_ID
When this value of the type **clockid_t** is used in a *clock*( ) or *timer**( ) function call, it is interpreted as the identifier of the CPU-time clock associated with the process making the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

function call.

TCT If _POSIX_THREAD_CPUTIME is defined, the following constant of the type **clockid_t** is defined in **<time.h>**:

CLOCK_THREAD_CPUTIME_ID
When this value of the type **clockid_t** is used in a *clock*( ) or *timer**( ) function call, it is interpreted as the identifier of the CPU-time clock associated with the thread making the function call.

2.9 Threads

THR The functionality described in this section is dependent on support of the Threads option (and the rest of this section is not further shaded for this option).

This section defines functionality to support multiple flows of control, called ‘‘threads’’, within a process. For the definition of threads, see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.393, Thread.

The specific functional areas covered by threads and their scope include:

- Thread management: the creation, control, and termination of multiple flows of control in the same process under the assumption of a common shared address space
- Synchronization primitives optimized for tightly coupled operation of multiple control flows in a common, shared address space

2.9.1 Thread-Safety

All functions defined by this volume of IEEE Std 1003.1-2001 shall be thread-safe, except that the following functions[1] need not be thread-safe.

asctime( )	*ecvt*( )	*gethostent*( )	*getutxline*( )	*putc_unlocked*( )
basename( )	*encrypt*( )	*getlogin*( )	*gmtime*( )	*putchar_unlocked*( )
catgets( )	*endgrent*( )	*getnetbyaddr*( )	*hcreate*( )	*putenv*( )
crypt( )	*endpwent*( )	*getnetbyname*( )	*hdestroy*( )	*pututxline*( )
ctime( )	*endutxent*( )	*getnetent*( )	*hsearch*( )	*rand*( )
dbm_clearerr( )	*fcvt*( )	*getopt*( )	*inet_ntoa*( )	*readdir*( )
dbm_close( )	*ftw*( )	*getprotobyname*( )	*l64a*( )	*setenv*( )
dbm_delete( )	*gcvt*( )	*getprotobynumber*( )	*lgamma*( )	*setgrent*( )
dbm_error( )	*getc_unlocked*( )	*getprotoent*( )	*lgammaf*( )	*setkey*( )
dbm_fetch( )	*getchar_unlocked*( )	*getpwent*( )	*lgammal*( )	*setpwent*( )
dbm_firstkey( )	*getdate*( )	*getpwnam*( )	*localeconv*( )	*setutxent*( )
dbm_nextkey( )	*getenv*( )	*getpwuid*( )	*localtime*( )	*strerror*( )
dbm_open( )	*getgrent*( )	*getservbyname*( )	*lrand48*( )	*strtok*( )
dbm_store( )	*getgrgid*( )	*getservbyport*( )	*mrand48*( )	*ttyname*( )
dirname( )	*getgrnam*( )	*getservent*( )	*nftw*( )	*unsetenv*( )

1. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

dlerror( )	*gethostbyaddr*( )	*getutxent*( )	*nl_langinfo*( )	*wcstombs*( )
drand48( )	*gethostbyname*( )	*getutxid*( )	*ptsname*( )	*wctomb*( )

The *ctermid*( ) and *tmpnam*( ) functions need not be thread-safe if passed a NULL argument. The *wcrtomb*( ) and *wcsrtombs*( ) functions need not be thread-safe if passed a NULL *ps* argument.

Implementations shall provide internal synchronization as necessary in order to satisfy this requirement.

2.9.2 Thread IDs

Although implementations may have thread IDs that are unique in a system, applications should only assume that thread IDs are usable and unique within a single process. The effect of calling any of the functions defined in this volume of IEEE Std 1003.1-2001 and passing as an argument the thread ID of a thread from another process is unspecified. A conforming implementation is free to reuse a thread ID after the thread terminates if it was created with the *detachstate* attribute set to PTHREAD_CREATE_DETACHED or if *pthread_detach*( ) or *pthread_join*( ) has been called for that thread. If a thread is detached, its thread ID is invalid for use as an argument in a call to *pthread_detach*( ) or *pthread_join*( ).

2.9.3 Thread Mutexes

A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources is determined by the scheduling policy.

A thread shall become the owner of a mutex, *m*, when one of the following occurs:

- It returns successfully from *pthread_mutex_lock*( ) with *m* as the *mutex* argument.
- It returns successfully from *pthread_mutex_trylock*( ) with *m* as the *mutex* argument.
- TMO It returns successfully from *pthread_mutex_timedlock*( ) with *m* as the mutex argument.
- It returns (successfully or not) from *pthread_cond_wait*( ) with *m* as the *mutex* argument (except as explicitly indicated otherwise for certain errors).
- It returns (successfully or not) from *pthread_cond_timedwait*( ) with *m* as the *mutex* argument (except as explicitly indicated otherwise for certain errors).

The thread shall remain the owner of *m* until one of the following occurs:

- It executes *pthread_mutex_unlock*( ) with *m* as the *mutex* argument
- It blocks in a call to *pthread_cond_wait*( ) with *m* as the *mutex* argument.
- It blocks in a call to *pthread_cond_timedwait*( ) with *m* as the *mutex* argument.

The implementation shall behave as if at all times there is at most one owner of any mutex.

A thread that becomes the owner of a mutex is said to have ‘‘acquired’’ the mutex and the mutex is said to have become ‘‘locked’’; when a thread gives up ownership of a mutex it is said to have ‘‘released’’ the mutex and the mutex is said to have become ‘‘unlocked’’.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.4 Thread Scheduling

TPS The functionality described in this section is dependent on support of the Thread Execution Scheduling option (and the rest of this section is not further shaded for this option).

Thread Scheduling Attributes

In support of the scheduling function, threads have attributes which are accessed through the **pthread_attr_t** thread creation attributes object.

The *contentionscope* attribute defines the scheduling contention scope of the thread to be either PTHREAD_SCOPE_PROCESS or PTHREAD_SCOPE_SYSTEM.

The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling attributes of the creating thread or to have its scheduling values set according to the other scheduling attributes in the **pthread_attr_t** object.

The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute defines the scheduling parameters for the thread. The interaction of threads having different policies within a process is described as part of the definition of those policies.

If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one of the priority-based policies defined under this option, the *schedparam* attribute contains the scheduling priority of the thread. A conforming implementation ensures that the priority value in *schedparam* is in the range associated with the scheduling policy when the thread attributes object is used to create a thread, or when the scheduling attributes of a thread are dynamically modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, the *schedparam* attribute supports four new members that are used for the sporadic server scheduling policy. These members are *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The meaning of these attributes is the same as in the definitions that appear under Section 2.8.4 (on page 44).

When a process is created, its single thread has a scheduling policy and associated attributes equal to the process' policy and attributes. The default scheduling contention scope value is implementation-defined. The default values of other scheduling attributes are implementation-defined.

Thread Scheduling Contention Scope

The scheduling contention scope of a thread defines the set of threads with which the thread competes for use of the processing resources. The scheduling operation selects at most one thread to execute on each processor at any point in time and the thread's scheduling attributes (for example, *priority*), whether under process scheduling contention scope or system scheduling contention scope, are the parameters used to determine the scheduling decision.

The scheduling contention scope, in the context of scheduling a mixed scope environment, affects threads as follows:

- A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends for resources with all other threads in the same scheduling allocation domain relative to their system scheduling attributes. The system scheduling attributes of a thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope are the scheduling attributes with which the thread was created. The system scheduling attributes of a thread created with PTHREAD_SCOPE_PROCESS scheduling contention scope are the implementation-defined mapping into system attribute space of the scheduling attributes with which the thread was created.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- Threads created with PTHREAD_SCOPE_PROCESS scheduling contention scope contend directly with other threads within their process that were created with PTHREAD_SCOPE_PROCESS scheduling contention scope. The contention is resolved based on the threads' scheduling attributes and policies. It is unspecified how such threads are scheduled relative to threads in other processes or threads with PTHREAD_SCOPE_SYSTEM scheduling contention scope.
- Conforming implementations shall support the PTHREAD_SCOPE_PROCESS scheduling contention scope, the PTHREAD_SCOPE_SYSTEM scheduling contention scope, or both.

Scheduling Allocation Domain

Implementations shall support scheduling allocation domains containing one or more processors. It should be noted that the presence of multiple processors does not automatically indicate a scheduling allocation domain size greater than one. Conforming implementations on multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation domains, and could define these scheduling allocation domains on a per-thread, per-process, or per-system basis, depending on the types of applications intended to be supported by the implementation. The scheduling allocation domain is independent of scheduling contention scope, as the scheduling contention scope merely defines the set of threads with which a thread contends for processor resources, while scheduling allocation domain defines the set of processors for which it contends. The semantics of how this contention is resolved among threads for processors is determined by the scheduling policies of the threads.

The choice of scheduling allocation domain size and the level of application control over scheduling allocation domains is implementation-defined. Conforming implementations may change the size of scheduling allocation domains and the binding of threads to scheduling allocation domains at any time.

For application threads with scheduling allocation domains of size equal to one, the scheduling rules defined for SCHED_FIFO and SCHED_RR shall be used; see **Scheduling Policies** (on page 44). All threads with system scheduling contention scope, regardless of the processes in which they reside, compete for the processor according to their priorities. Threads with process scheduling contention scope compete only with other threads with process scheduling contention scope within their process.

For application threads with scheduling allocation domains of size greater than one, the rules TSP defined for SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC shall be used in an implementation-defined manner. Each thread with system scheduling contention scope competes for the processors in its scheduling allocation domain in an implementation-defined manner according to its priority. Threads with process scheduling contention scope are scheduled relative to other threads within the same scheduling contention scope in the process.

TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, the rules defined for SCHED_SPORADIC in **Scheduling Policies** (on page 44) shall be used in an implementation-defined manner for application threads whose scheduling allocation domain size is greater than one.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Scheduling Documentation

If _POSIX_PRIORITY_SCHEDULING is defined, then any scheduling policies beyond TSP SCHED_OTHER, SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC, as well as the effects of the scheduling policies indicated by these other values, and the attributes required in order to support such a policy, are implementation-defined. Furthermore, the implementation shall document the effect of all processor scheduling allocation domain values supported for these policies.

2.9.5 Thread Cancellation

The thread cancellation mechanism allows a thread to terminate the execution of any other thread in the process in a controlled manner. The target thread (that is, the one that is being canceled) is allowed to hold cancellation requests pending in a number of ways and to perform application-specific cleanup processing when the notice of cancellation is acted upon.

Cancellation is controlled by the cancellation control functions. Each thread maintains its own cancelability state. Cancellation may only occur at cancellation points or when the thread is asynchronously cancelable.

The thread cancellation mechanism described in this section depends upon programs having set *deferred* cancelability state, which is specified as the default. Applications shall also carefully follow static lexical scoping rules in their execution behavior. For example, use of *setjmp*( ), *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary scope pop operation results in undefined behavior.

Use of asynchronous cancelability while holding resources which potentially need to be released may result in resource loss. Similarly, cancellation scopes may only be safely manipulated (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

2.9.5.1 Cancelability States

The cancelability state of a thread determines the action taken upon receipt of a cancellation request. The thread may control cancellation in a number of ways.

Each thread maintains its own cancelability state, which may be encoded in two bits:

1. Cancelability-Enable: When cancelability is PTHREAD_CANCEL_DISABLE (as defined in the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**), cancellation requests against the target thread are held pending. By default, cancelability is set to PTHREAD_CANCEL_ENABLE (as defined in **<pthread.h>**).

2. Cancelability Type: When cancelability is enabled and the cancelability type is PTHREAD_CANCEL_ASYNCHRONOUS (as defined in **<pthread.h>**), new or pending cancellation requests may be acted upon at any time. When cancelability is enabled and the cancelability type is PTHREAD_CANCEL_DEFERRED (as defined in **<pthread.h>**), cancellation requests are held pending until a cancellation point (see below) is reached. If cancelability is disabled, the setting of the cancelability type has no immediate effect as all cancellation requests are held pending; however, once cancelability is enabled again the new type is in effect. The cancelability type is PTHREAD_CANCEL_DEFERRED in all newly created threads including the thread in which *main*( ) was first invoked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.9.5.2 Cancellation Points

Cancellation points shall occur when a thread is executing the following functions:

accept( )	*mq_timedsend*( )	*putmsg*( )	*sigpause*( )
aio_suspend( )	*msgrcv*( )	*putpmsg*( )	*sigsuspend*( )
clock_nanosleep( )	*msgsnd*( )	*pwrite*( )	*sigtimedwait*( )
close( )	*msync*( )	*read*( )	*sigwait*( )
connect( )	*nanosleep*( )	*readv*( )	*sigwaitinfo*( )
creat( )	*open*( )	*recv*( )	*sleep*( )
fcntl( )[2]	*pause*( )	*recvfrom*( )	*system*( )
fsync( )	*poll*( )	*recvmsg*( )	*tcdrain*( )
getmsg( )	*pread*( )	*select*( )	*usleep*( )
getpmsg( )	*pselect*( )	*sem_timedwait*( )	*wait*( )
lockf( )	*pthread_cond_timedwait*( )	*sem_wait*( )	*waitid*( )
mq_receive( )	*pthread_cond_wait*( )	*send*( )	*waitpid*( )
mq_send( )	*pthread_join*( )	*sendmsg*( )	*write*( )
mq_timedreceive( )	*pthread_testcancel*( )	*sendto*( )	*writev*( )

---

2. When the *cmd* argument is F_SETLKW.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A cancellation point may also occur when a thread is executing the following functions:

catclose( )
catgets( )
catopen( )
closedir( )
closelog( )
ctermid( )
dbm_close( )
dbm_delete( )
dbm_fetch( )
dbm_nextkey( )
dbm_open( )
dbm_store( )
dlclose( )
dlopen( )
endgrent( )
endhostent( )
endnetent( )
endprotoent( )
endpwent( )
endservent( )
endutxent( )
fclose( )
fcntl( )[3]
fflush( )
fgetc( )
fgetpos( )
fgets( )
fgetwc( )
fgetws( )
fopen( )
fprintf( )
fputc( )
fputs( )
fputwc( )
fputws( )
fread( )
freopen( )
fscanf( )
fseek( )
fseeko( )
fsetpos( )
ftell( )
ftello( )
ftw( )
fwprintf( )
fwrite( )
fwscanf( )
getc( )
getc_unlocked( )
getchar( )
getchar_unlocked( )
getcwd( )
getdate( )
getgrent( )
getgrgid( )
getgrgid_r( )
getgrnam( )
getgrnam_r( )
gethostbyaddr( )
gethostbyname( )
gethostent( )
gethostname( )
getlogin( )
getlogin_r( )
getnetbyaddr( )
getnetbyname( )
getnetent( )
getprotobyname( )
getprotobynumber( )
getprotoent( )
getpwent( )
getpwnam( )
getpwnam_r( )
getpwuid( )
getpwuid_r( )
gets( )
getservbyname( )
getservbyport( )
getservent( )
getutxent( )
getutxid( )
getutxline( )
getwc( )
getwchar( )
getwd( )
glob( )
iconv_close( )
iconv_open( )
ioctl( )
lseek( )
mkstemp( )
nftw( )
opendir( )
openlog( )
pclose( )
perror( )
popen( )
posix_fadvise( )
posix_fallocate( )
posix_madvise( )
posix_spawn( )
posix_spawnp( )
posix_trace_clear( )
posix_trace_close( )
posix_trace_create( )
posix_trace_create_withlog( )
posix_trace_eventtypelist_getnext_id( )
posix_trace_eventtypelist_rewind( )
posix_trace_flush( )
posix_trace_get_attr( )
posix_trace_get_filter( )
posix_trace_get_status( )
posix_trace_getnext_event( )
posix_trace_open( )
posix_trace_rewind( )
posix_trace_set_filter( )
posix_trace_shutdown( )
posix_trace_timedgetnext_event( )
posix_typed_mem_open( )
printf( )
pthread_rwlock_rdlock( )
pthread_rwlock_timedrdlock( )
pthread_rwlock_timedwrlock( )
pthread_rwlock_wrlock( )
putc( )
putc_unlocked( )
putchar( )
putchar_unlocked( )
puts( )
pututxline( )
putwc( )
putwchar( )
readdir( )
readdir_r( )
remove( )
rename( )
rewind( )
rewinddir( )
scanf( )
seekdir( )
semop( )
setgrent( )
sethostent( )
setnetent( )
setprotoent( )
setpwent( )
setservent( )
setutxent( )
strerror( )
syslog( )
tmpfile( )
tmpnam( )
ttyname( )
ttyname_r( )
ungetc( )
ungetwc( )
unlink( )
vfprintf( )
vfwprintf( )
vprintf( )
vwprintf( )
wprintf( )
wscanf( )

An implementation shall not introduce cancellation points into any other functions specified in this volume of IEEE Std 1003.1-2001.

3. For any value of the *cmd* argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The side effects of acting upon a cancellation request while suspended during a call of a function are the same as the side effects that may be seen in a single-threaded program when a call to a function is interrupted by a signal and the given function returns [EINTR]. Any such side effects occur before any cancellation cleanup handlers are called.

Whenever a thread has cancelability enabled and a cancellation request has been made with that thread as the target, and the thread then calls any function that is a cancellation point (such as *pthread_testcancel*( ) or *read*( )), the cancellation request shall be acted upon before the function returns. If a thread has cancelability enabled and a cancellation request is made with the thread as a target while the thread is suspended at a cancellation point, the thread shall be awakened and the cancellation request shall be acted upon. However, if the thread is suspended at a cancellation point and the event for which it is waiting occurs before the cancellation request is acted upon, it is unspecified whether the cancellation request is acted upon or whether the cancellation request remains pending and the thread resumes normal execution.

2.9.5.3 *Thread Cancellation Cleanup Handlers*

Each thread maintains a list of cancellation cleanup handlers. The programmer uses the *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ) functions to place routines on and remove routines from this list.

When a cancellation request is acted upon, the routines in the list are invoked one by one in LIFO sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First Out). The thread invokes the cancellation cleanup handler with cancellation disabled until the last cancellation cleanup handler returns. When the cancellation cleanup handler for a scope is invoked, the storage for that scope remains valid. If the last cancellation cleanup handler returns, thread execution is terminated and a status of PTHREAD_CANCELED is made available to any threads joining with the target. The symbolic constant PTHREAD_CANCELED expands to a constant expression of type (**void ***) whose value matches no pointer to an object in memory nor the value NULL.

The cancellation cleanup handlers are also invoked when the thread calls *pthread_exit*( ).

A side effect of acting upon a cancellation request while in a condition variable wait is that the mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread is no longer considered to be waiting for the condition and the thread shall not have consumed any pending condition signals on the condition.

A cancellation cleanup handler cannot exit via *longjmp*( ) or *siglongjmp*( ).

2.9.5.4 *Async-Cancel Safety*

The *pthread_cancel*( ), *pthread_setcancelstate*( ), and *pthread_setcanceltype*( ) functions are defined to be async-cancel safe.

No other functions in this volume of IEEE Std 1003.1-2001 are required to be async-cancel-safe.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.6 Thread Read-Write Locks

Multiple readers, single writer (read-write) locks allow many threads to have simultaneous read-only access to data while allowing only one thread to have exclusive write access at any given time. They are typically used to protect data that is read more frequently than it is changed.

One or more readers acquire read access to the resource by performing a read lock operation on the associated read-write lock. A writer acquires exclusive write access by performing a write lock operation. Basically, all readers exclude any writers and a writer excludes all readers and any other writers.

A thread that has blocked on a read-write lock (for example, has not yet returned from a *pthread_rwlock_rdlock*( ) or *pthread_rwlock_wrlock*( ) call) shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources shall be determined by the scheduling policy.

Read-write locks can be used to synchronize threads in the current process and other processes if they are allocated in memory that is writable and shared among the cooperating processes and have been initialized for this behavior.

2.9.7 Thread Interactions with Regular File Operations

All of the functions *chmod*( ), *close*( ), *fchmod*( ), *fcntl*( ), *fstat*( ), *ftruncate*( ), *lseek*( ), *open*( ), *read*( ), *readlink*( ), *stat*( ), *symlink*( ), and *write*( ) shall be atomic with respect to each other in the effects specified in IEEE Std 1003.1-2001 when they operate on regular files. If two threads each call one of these functions, each call shall either see all of the specified effects of the other call, or none of them.

2.10 Sockets

A socket is an endpoint for communication using the facilities described in this section. A socket is created with a specific socket type, described in Section 2.10.6 (on page 59), and is associated with a specific protocol, detailed in Section 2.10.3 (on page 59). A socket is accessed via a file descriptor obtained when the socket is created.

2.10.1 Address Families

All network protocols are associated with a specific address family. An address family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. An address family is normally comprised of a number of protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not required that an address family support all socket types. An address family may contain multiple protocols supporting the same socket abstraction.

Section 2.10.17 (on page 66), Section 2.10.19 (on page 67), and Section 2.10.20 (on page 67), respectively, describe the use of sockets for local UNIX connections, for Internet protocols based on IPv4, and for Internet protocols based on IPv6.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

2.10.2 Addressing

An address family defines the format of a socket address. All network addresses are described using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**. However, each address family imposes finer and more specific structure, generally defining a structure with fields specific to the address family. The field *sa_family* in the **sockaddr** structure contains the address family identifier, specifying the format of the *sa_data* area. The size of the *sa_data* area is unspecified.

2.10.3 Protocols

A protocol supports one of the socket abstractions detailed in Section 2.10.6. Selecting a protocol involves specifying the address family, socket type, and protocol number to the *socket*( ) function. Certain semantics of the basic socket abstractions are protocol-specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism.

2.10.4 Routing

Sockets provides packet routing facilities. A routing information database is maintained, which is used in selecting the appropriate network interface when transmitting packets.

2.10.5 Interfaces

Each network interface in a system corresponds to a path through which messages can be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, do not.

2.10.6 Socket Types

A socket is created with a specific type, which defines the communication semantics and which RS allows the selection of an appropriate communication protocol. Four types are defined: SOCK_RAW, SOCK_STREAM, SOCK_SEQPACKET, and SOCK_DGRAM. Implementations may specify additional socket types.

The SOCK_STREAM socket type provides reliable, sequenced, full-duplex octet streams between the socket and a peer to which the socket is connected. A socket of type SOCK_STREAM must be in a connected state before any data may be sent or received. Record boundaries are not maintained; data sent on a stream socket using output operations of one size may be received using input operations of smaller or larger sizes without loss of data. Data may be buffered; successful return from an output function does not imply that the data has been delivered to the peer or even transmitted from the local system. If data cannot be successfully transmitted within a given time then the connection is considered broken, and subsequent operations shall fail. A SIGPIPE signal is raised if a thread sends on a broken stream (one that is no longer connected). Support for an out-of-band data transmission facility is protocol-specific.

The SOCK_SEQPACKET socket type is similar to the SOCK_STREAM type, and is also connection-oriented. The only difference between these types is that record boundaries are maintained using the SOCK_SEQPACKET type. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers parts of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag in the received message flags returned by the *recvmsg*( ) function. It is protocol-specific whether a maximum record size is imposed.

The SOCK_DGRAM socket type supports connectionless data transfer which is not necessarily acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

broadcast) in each output operation, and incoming datagrams may be received from multiple sources. The source address of each datagram is available when receiving the datagram. An application may also pre-specify a peer address, in which case calls to output functions shall send to the pre-specified peer. If a peer has been specified, only datagrams from that peer shall be received. A datagram must be sent in a single output operation, and must be received in a single input operation. The maximum size of a datagram is protocol-specific; with some protocols, the limit is implementation-defined. Output datagrams may be buffered within the system; thus, a successful return from an output function does not guarantee that a datagram is actually sent or received. However, implementations should attempt to detect any errors possible before the return of an output function, reporting any error by an unsuccessful return value.

RS The SOCK_RAW socket type is similar to the SOCK_DGRAM type. It differs in that it is normally used with communication providers that underlie those used for the other socket types. For this reason, the creation of a socket with type SOCK_RAW shall require appropriate privilege. The format of datagrams sent and received with this socket type generally include specific protocol headers, and the formats are protocol-specific and implementation-defined.

2.10.7 Socket I/O Mode

The I/O mode of a socket is described by the O_NONBLOCK file status flag which pertains to the open file description for the socket. This flag is initially off when a socket is created, but may be set and cleared by the use of the F_SETFL command of the *fcntl*( ) function.

When the O_NONBLOCK flag is set, functions that would normally block until they are complete shall either return immediately with an error, or shall complete asynchronously to the execution of the calling process. Data transfer operations (the *read*( ), *write*( ), *send*( ), and *recv*( ) functions) shall complete immediately, transfer only as much as is available, and then return without blocking, or return an error indicating that no transfer could be made without blocking. The *connect*( ) function initiates a connection and shall return without blocking when O_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection was initiated successfully, but that it has not yet completed.

2.10.8 Socket Owner

The owner of a socket is unset when a socket is created. The owner may be set to a process ID or process group ID using the F_SETOWN command of the *fcntl*( ) function.

2.10.9 Socket Queue Limits

The transmit and receive queue sizes for a socket are set when the socket is created. The default sizes used are both protocol-specific and implementation-defined. The sizes may be changed using the *setsockopt*( ) function.

2.10.10 Pending Error

Errors may occur asynchronously, and be reported to the socket in response to input from the network protocol. The socket stores the pending error to be reported to a user of the socket at the next opportunity. The error is returned in response to a subsequent *send*( ), *recv*( ), or *getsockopt*( ) operation on the socket, and the pending error is then cleared.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.11 Socket Receive Queue

A socket has a receive queue that buffers data when it is received by the system until it is removed by a receive call. Depending on the type of the socket and the communication provider, the receive queue may also contain ancillary data such as the addressing and other protocol data associated with the normal data in the queue, and may contain out-of-band or expedited data. The limit on the queue size includes any normal, out-of-band data, datagram source addresses, and ancillary data in the queue. The description in this section applies to all sockets, even though some elements cannot be present in some instances.

The contents of a receive buffer are logically structured as a series of data segments with associated ancillary data and other information. A data segment may contain normal data or out-of-band data, but never both. A data segment may complete a record if the protocol supports records (always true for types SOCK_SEQPACKET and SOCK_DGRAM). A record may be stored as more than one segment; the complete record might never be present in the receive buffer at one time, as a portion might already have been returned to the application, and another portion might not yet have been received from the communications provider. A data segment may contain ancillary protocol data, which is logically associated with the segment. Ancillary data is received as if it were queued along with the first normal data octet in the segment (if any). A segment may contain ancillary data only, with no normal or out-of-band data. For the purposes of this section, a datagram is considered to be a data segment that terminates a record, and that includes a source address as a special type of ancillary data. Data segments are placed into the queue as data is delivered to the socket by the protocol. Normal data segments are placed at the end of the queue as they are delivered. If a new segment contains the same type of data as the preceding segment and includes no ancillary data, and if the preceding segment does not terminate a record, the segments are logically merged into a single segment.

The receive queue is logically terminated if an end-of-file indication has been received or a connection has been terminated. A segment shall be considered to be terminated if another segment follows it in the queue, if the segment completes a record, or if an end-of-file or other connection termination has been reported. The last segment in the receive queue shall also be considered to be terminated while the socket has a pending error to be reported.

A receive operation shall never return data or ancillary data from more than one segment.

2.10.12 Socket Out-of-Band Data State

The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed in the socket receive queue, either at the end of the queue or before all normal data in the queue. In this case, out-of-band data is returned to an application program by a normal receive call. Out-of-band data may also be queued separately rather than being placed in the socket receive queue, in which case it shall be returned only in response to a receive call that requests out-of-band data. It is protocol-specific whether an out-of-band data mark is placed in the receive queue to demarcate data preceding the out-of-band data and following the out-of-band data. An out-of-band data mark is logically an empty data segment that cannot be merged with other segments in the queue. An out-of-band data mark is never returned in response to an input operation. The *sockatmark*( ) function can be used to test whether an out-of-band data mark is the first element in the queue. If an out-of-band data mark is the first element in the queue when an input function is called without the MSG_PEEK option, the mark is removed from the queue and the following data (if any) is processed as if the mark had not been present.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.13 Connection Indication Queue

Sockets that are used to accept incoming connections maintain a queue of outstanding connection indications. This queue is a list of connections that are awaiting acceptance by the application; see *listen*( ).

2.10.14 Signals

One category of event at the socket interface is the generation of signals. These signals report protocol events or process errors relating to the state of the socket. The generation or delivery of a signal does not change the state of the socket, although the generation of the signal may have been caused by a state change.

The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no longer able to send. In addition, the send operation fails with the error [EPIPE].

If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the status of the socket is specified in Section 2.10.17 (on page 66), Section 2.10.19 (on page 67), and Section 2.10.20 (on page 67). Depending on the protocol, the expedited data may or may not have arrived at the time of signal generation.

2.10.15 Asynchronous Errors

If any of the following conditions occur asynchronously for a socket, the corresponding value listed below shall become the pending error for the socket:

[ECONNABORTED]
The connection was aborted locally.

[ECONNREFUSED]
For a connection-mode socket attempting a non-blocking connection, the attempt to connect was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram was forcefully rejected.

[ECONNRESET]
The peer has aborted the connection.

[EHOSTDOWN]
The destination host has been determined to be down or disconnected.

[EHOSTUNREACH]
The destination host is not reachable.

[EMSGSIZE]
For a connectionless-mode socket, the size of a previously sent datagram prevented delivery.

[ENETDOWN]
The local network connection is not operational.

[ENETRESET]
The connection was aborted by the network.

[ENETUNREACH]
The destination network is not reachable.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.10.16 Use of Options

There are a number of socket options which either specialize the behavior of a socket or provide useful information. These options may be set at different protocol levels and are always present at the uppermost ‘‘socket’’ level.

Socket options are manipulated by two functions, *getsockopt*( ) and *setsockopt*( ). These functions allow an application program to customize the behavior and characteristics of a socket to provide the desired effect.

All of the options have default values. The type and meaning of these values is defined by the protocol level to which they apply. Instead of using the default values, an application program may choose to customize one or more of the options. However, in the bulk of cases, the default values are sufficient for the application.

Some of the options are used to enable or disable certain behavior within the protocol modules (for example, turn on debugging) while others may be used to set protocol-specific information (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is introduced, its effect on the underlying protocol modules is described.

Table 2-1 shows the value for the socket level.

Table 2-1 Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

Table 2-2 (on page 64) lists those options present at the socket level; that is, when the *level* parameter of the *getsockopt*( ) or *setsockopt*( ) function is SOL_SOCKET, the types of the option value parameters associated with each option, and a brief synopsis of the meaning of the option value parameter. Unless otherwise noted, each may be examined with *getsockopt*( ) and set with *setsockopt*( ) on all types of socket.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 2-2 Socket-Level Options

Option	Parameter Type	Parameter Meaning
SO_BROADCAST	**int**	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	**int**	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	**int**	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	**int**	Requests and clears pending error information on the socket (*getsockopt*( ) only).
SO_KEEPALIVE	**int**	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	**struct linger**	Specify actions to be taken for queued, unsent data on *close*( ): linger on/off and linger time in seconds.
SO_OOBINLINE	**int**	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	**int**	Size of receive buffer (in bytes).
SO_RCVLOWAT	**int**	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	**struct timeval**	Timeout value for a socket receive operation.
SO_REUSEADDR	**int**	Non-zero requests reuse of local addresses in *bind*( ) (protocol-specific).
SO_SNDBUF	**int**	Size of send buffer (in bytes).
SO_SNDLOWAT	**int**	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	**struct timeval**	Timeout value for a socket send operation.
SO_TYPE	**int**	Identify socket type (*getsockopt*( ) only).

The SO_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

The SO_DEBUG option enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO_DEBUG is for debugging to be turned off.

The SO_DONTROUTE option requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. It is protocol-specific whether this option has any effect and how the outgoing network interface is chosen. Support for this option with each protocol is implementation-defined.

The SO_ERROR option is used only on *getsockopt*( ). When this option is specified, *getsockopt*( ) shall return any pending error on the socket and clear the error status. It shall return a value of 0 if there is no pending error. SO_ERROR may be used to check for asynchronous errors on connected connectionless-mode sockets or for other types of asynchronous errors. SO_ERROR has no default value.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The SO_KEEPALIVE option enables the periodic transmission of messages on a connected socket. The behavior of this option is protocol-specific. The default value for SO_KEEPALIVE is zero, specifying that this capability is turned off.

The SO_LINGER option controls the action of the interface when unsent messages are queued on a socket and a *close*( ) is performed. The details of this option are protocol-specific. The default value for SO_LINGER is zero, or off, for the *l_onoff* element of the option value and zero seconds for the linger time specified by the *l_linger* element.

The SO_OOBINLINE option is valid only on protocols that support out-of-band data. The SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it is then accessible using the *read*( ) or *recv*( ) functions without the MSG_OOB flag set. The default for SO_OOBINLINE is off; that is, for out-of-band data not to be placed in the normal data input queue.

The SO_RCVBUF option requests that the buffer space allocated for receive operations on this socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer size for high volume connections, or may decrease buffer size to limit the possible backlog of incoming data. The default value for the SO_RCVBUF option value is implementation-defined, and may vary by protocol.

The maximum value for the option for a socket may be obtained by the use of the *fpathconf*( ) function, using the value _PC_SOCK_MAXBUF.

The SO_RCVLOWAT option sets the minimum number of bytes to process for socket input operations. In general, receive calls block until any (non-zero) amount of data is received, then return the smaller of the amount available or the amount requested. The default value for SO_RCVLOWAT is 1, and does not affect the general case. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. Receive calls may still return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned (for example, out-of-band data). As mentioned previously, the default value for SO_RCVLOWAT is 1 byte. It is implementation-defined whether the SO_RCVLOWAT option can be set.

The SO_RCVTIMEO option is an option to set a timeout value for input operations. It accepts a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or *errno* shall be set to [EWOULDBLOCK] if no data were received. The default for this option is the value zero, which indicates that a receive operation will not time out. It is implementation-defined whether the SO_RCVTIMEO option can be set.

The SO_REUSEADDR option indicates that the rules used in validating addresses supplied in a *bind*( ) should allow reuse of local addresses. Operation of this option is protocol-specific. The default value for SO_REUSEADDR is off; that is, reuse of local addresses is not permitted.

The SO_SNDBUF option requests that the buffer space allocated for send operations on this socket be set to the value, in bytes, of the option value. The default value for the SO_SNDBUF option value is implementation-defined, and may vary by protocol. The maximum value for the option for a socket may be obtained by the use of the *fpathconf*( ) function, using the value _PC_SOCK_MAXBUF.

The SO_SNDLOWAT option sets the minimum number of bytes to process for socket output operations. Most output operations process all of the data supplied by the call, delivering data to the protocol for transmission and blocking as necessary for flow control. Non-blocking output operations process as much data as permitted subject to flow control without blocking, but

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. A *select*( ) operation testing the ability to write to a socket shall return true only if the send low water mark could be processed. The default value for SO_SNDLOWAT is implementation-defined and protocol-specific. It is implementation-defined whether the SO_SNDLOWAT option can be set.

The SO_SNDTIMEO option is an option to set a timeout value for the amount of time that an output function shall block because flow control prevents data from being sent. As noted in Table 2-2 (on page 64), the option value is a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an output operation to complete. If a send operation has blocked for this much time, it shall return with a partial count or *errno* set to [EWOULDBLOCK] if no data were sent. The default for this option is the value zero, which indicates that a send operation will not time out. It is implementation-defined whether the SO_SNDTIMEO option can be set.

The SO_TYPE option is used only on *getsockopt*( ). When this option is specified, *getsockopt*( ) shall return the type of the socket (for example, SOCK_STREAM). This option is useful to servers that inherit sockets on start-up. SO_TYPE has no default value.

2.10.17 Use of Sockets for Local UNIX Connections

Support for UNIX domain sockets is mandatory.

UNIX domain sockets provide process-to-process communication in a single system.

2.10.17.1 Headers

The symbolic constant AF_UNIX defined in the **<sys/socket.h>** header is used to identify the UNIX domain address family. The **<sys/un.h>** header contains other definitions used in connection with UNIX domain sockets. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

The **sockaddr_storage** structure defined in **<sys/socket.h>** shall be large enough to accommodate a **sockaddr_un** structure (see the **<sys/un.h>** header defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_un** structures and used to access the fields of those structures without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_un** structure, the *ss_family* field maps onto the *sun_family* field.

2.10.18 Use of Sockets over Internet Protocols

When a socket is created in the Internet family with a protocol value of zero, the implementation shall use the protocol listed below for the type of socket created.

SOCK_STREAM	IPPROTO_TCP.
SOCK_DGRAM	IPPROTO_UDP.
RS SOCK_RAW	IPPROTO_RAW.
SOCK_SEQPACKET	Unspecified.

RS A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The default protocol for type SOCK_RAW shall be identified in the IP header with the value IPPROTO_RAW. Applications should not use the default protocol when creating a socket with type SOCK_RAW, but should identify a specific protocol by value. The ICMP control protocol is accessible from a raw socket by specifying a value of IPPROTO_ICMP for protocol.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.19 Use of Sockets over Internet Protocols Based on IPv4

Support for sockets over Internet protocols based on IPv4 is mandatory.

2.10.19.1 Headers

The symbolic constant AF_INET defined in the **<sys/socket.h>** header is used to identify the IPv4 Internet address family. The **<netinet/in.h>** header contains other definitions used in connection with IPv4 Internet sockets. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

The **sockaddr_storage** structure defined in **<sys/socket.h>** shall be large enough to accommodate a **sockaddr_in** structure (see the **<netinet/in.h>** header defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_in** structures and used to access the fields of those structures without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_in** structure, the *ss_family* field maps onto the *sin_family* field.

2.10.20 Use of Sockets over Internet Protocols Based on IPv6

IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. This functionality is dependent on support of the IPV6 option (and the rest of this section is not further shaded for this option).

To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 68).

2.10.20.1 Addressing

IPv6 overcomes the addressing limitations of previous versions by using 128-bit addresses instead of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

There are three kinds of IPv6 address:

Unicast
: Identifies a single interface.

  A unicast address can be global, link-local (designed for use on a single link), or site-local (designed for systems not connected to the Internet). Link-local and site-local addresses need not be globally unique.

Anycast
: Identifies a set of interfaces such that a packet sent to the address can be delivered to any member of the set.

  An anycast address is similar to a unicast address; the nodes to which an anycast address is assigned must be explicitly configured to know that it is an anycast address.

Multicast
: Identifies a set of interfaces such that a packet sent to the address should be delivered to every member of the set.

  An application can send multicast datagrams by simply specifying an IPv6 multicast address in the *address* argument of *sendto*( ). To receive multicast datagrams, an application must join the multicast group (using *setsockopt*( ) with IPV6_JOIN_GROUP) and must bind to the socket the UDP port on which datagrams will be received. Some applications should also bind the multicast group address to the socket, to prevent other datagrams destined to that port from being delivered to the socket.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A multicast address can be global, node-local, link-local, site-local, or organization-local.

The following special IPv6 addresses are defined:

Unspecified
: An address that is not assigned to any interface and is used to indicate the absence of an address.

Loopback
: A unicast address that is not assigned to any interface and can be used by a node to send packets to itself.

Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:

IPv4-compatible addresses
: These are assigned to nodes that support IPv6 and can be used when traffic is ‘‘tunneled’’ through IPv4.

IPv4-mapped addresses
: These are used to represent IPv4 addresses in IPv6 address format; see Section 2.10.20.2.

Note that the unspecified address and the loopback address must not be treated as IPv4-compatible addresses.

2.10.20.2 Compatibility with IPv4

The API provides the ability for IPv6 applications to interoperate with applications using IPv4, by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the *getaddrinfo*( ) function when the specified host has only IPv4 addresses.

Applications can use AF_INET6 sockets to open TCP connections to IPv4 nodes, or send UDP packets to IPv4 nodes, by simply encoding the destination’s IPv4 address as an IPv4-mapped IPv6 address, and passing that address, within a **sockaddr_in6** structure, in the *connect*( ), *sendto*( ), or *sendmsg*( ) function. When applications use AF_INET6 sockets to accept TCP connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return the peer’s address to the application in the *accept*( ), *recvfrom*( ), *recvmsg*( ), or *getpeername*( ) function using a **sockaddr_in6** structure encoded this way. If a node has an IPv4 address, then the implementation shall allow applications to communicate using that address via an AF_INET6 socket. In such a case, the address will be represented at the API by the corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an AF_INET6 socket bound to **in6addr_any** to receive inbound connections and packets destined to one of the node’s IPv4 addresses.

An application can use AF_INET6 sockets to bind to a node’s IPv4 address by specifying the address as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure in the *bind*( ) function. For an AF_INET6 socket bound to a node’s IPv4 address, the system shall return the address in the *getsockname*( ) function as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure.

2.10.20.3 Interface Identification

Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1; zero is not used. There may be gaps so that there is no current interface for a particular positive index. Each interface also has a unique implementation-defined name.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2810 *2.10.20.4 Options*

2811 The following options apply at the IPPROTO_IPV6 level:

2812 IPV6_JOIN_GROUP
2813 When set via *setsockopt*( ), it joins the application to a multicast group on an interface
2814 (identified by its index) and addressed by a given multicast address, enabling packets sent
2815 to that address to be read via the socket. If the interface index is specified as zero, the
2816 system selects the interface (for example, by looking up the address in a routing table and
2817 using the resulting interface).

2818 An attempt to read this option using *getsockopt*( ) shall result in an [EOPNOTSUPP] error.

2819 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

2820 IPV6_LEAVE_GROUP
2821 When set via *setsockopt*( ), it removes the application from the multicast group on an
2822 interface (identified by its index) and addressed by a given multicast address.

2823 An attempt to read this option using *getsockopt*( ) shall result in an [EOPNOTSUPP] error.

2824 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

2825 IPV6_MULTICAST_HOPS
2826 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the
2827 socket. Its possible values are the same as those of IPV6_UNICAST_HOPS. If the
2828 IPV6_MULTICAST_HOPS option is not set, a value of 1 is assumed. This option can be set
2829 via *setsockopt*( ) and read via *getsockopt*( ).

2830 The parameter type of this option is a pointer to an **int**. (Default value: 1)

2831 IPV6_MULTICAST_IF
2832 The index of the interface to be used for outgoing multicast packets. It can be set via
2833 *setsockopt*( ) and read via *getsockopt*( ). If the interface index is specified as zero, the system
2834 selects the interface (for example, by looking up the address in a routing table and using the
2835 resulting interface).

2836 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)

2837 IPV6_MULTICAST_LOOP
2838 This option controls whether outgoing multicast packets should be delivered back to the
2839 local application when the sending interface is itself a member of the destination multicast
2840 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an
2841 [EINVAL] error. This option can be set via *setsockopt*( ) and read via *getsockopt*( ).

2842 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean
2843 value. (Default value: 1)

2844 IPV6_UNICAST_HOPS
2845 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the
2846 socket. If the option is not set, or is set to –1, the system selects a default value. Attempts to
2847 set a value less than –1 or greater than 255 shall result in an [EINVAL] error. This option can
2848 be set via *setsockopt*( ) and read via *getsockopt*( ).

2849 The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

2850 IPV6_V6ONLY
2851 This socket option restricts AF_INET6 sockets to IPv6 communications only. AF_INET6
2852 sockets may be used for both IPv4 and IPv6 communications. Some applications may want
2853 to restrict their use of an AF_INET6 socket to IPv6 communications only. For these

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

applications, the IPv6_V6ONLY socket option is defined. When this option is turned on, the socket can be used to send and receive IPv6 packets only. This is an IPPROTO_IPV6-level option.

The parameter type of this option is a pointer to an **int** which is used as a Boolean value. (Default value: 0)

An [EOPNOTSUPP] error shall result if IPV6_JOIN_GROUP or IPV6_LEAVE_GROUP is used with *getsockopt*( ).

2.10.20.5 Headers

The symbolic constant AF_INET6 is defined in the **<sys/socket.h>** header to identify the IPv6 Internet address family. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

The **sockaddr_storage** structure defined in **<sys/socket.h>** shall be large enough to accommodate a **sockaddr_in6** structure (see the **<netinet/in.h>** header defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_in6** structures and used to access the fields of those structures without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_in6** structure, the *ss_family* field maps onto the *sin6_family* field.

The **<netinet/in.h>**, **<arpa/inet.h>**, and **<netdb.h>** headers contain other definitions used in connection with IPv6 Internet sockets; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

2.11 Tracing

TRC This section describes extensions to support tracing of user applications. This functionality is dependent on support of the Trace option (and the rest of this section is not further shaded for this option).

The tracing facilities defined in IEEE Std 1003.1-2001 allow a process to select a set of trace event types, to activate a trace stream of the selected trace events as they occur in the flow of execution, and to retrieve the recorded trace events.

The tracing operation relies on three logically different components: the traced process, the controller process, and the analyzer process. During the execution of the traced process, when a trace point is reached, a trace event is recorded into the trace streams created for that process in which the associated trace event type identifier is not being filtered out. The controller process controls the operation of recording the trace events into the trace stream. It shall be able to:

- Initialize the attributes of a trace stream
- Create the trace stream (for a specified traced process) using those attributes
- Start and stop tracing for the trace stream
- Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- Shut a trace stream down

These operations can be done for an active trace stream. The analyzer process retrieves the traced events either at runtime, when the trace stream has not yet been shut down, but is still recording trace events; or after opening a trace log that had been previously recorded and shut down. These three logically different operations can be performed by the same process, or can be

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

distributed into different processes.

A trace stream identifier can be created by a call to *posix_trace_create*(), *posix_trace_create_withlog*(), or *posix_trace_open*(). The *posix_trace_create*() and *posix_trace_create_withlog*() functions should be used by a controller process. The *posix_trace_open*() should be used by an analyzer process.

The tracing functions can serve different purposes. One purpose is debugging the possibly pre-instrumented code, while another is post-mortem fault analysis. These two potential uses differ in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and permits focusing on expected information; while the second needs comprehensive trace capabilities in order to be able to record all types of information.

The events to be traced belong to two classes:

1. User trace events (generated by the application instrumentation)
2. System trace events (generated by the operating system)

The trace interface defines several system trace event types associated with control of and operation of the trace stream. This small set of system trace events includes the minimum required to interpret correctly the trace event information present in the stream. Other desirable system trace events for some particular application profile may be implemented and are encouraged; for example, process and thread scheduling, signal occurrence, and so on.

Each traced process shall have a mapping of the trace event names to trace event type identifiers that have been defined for that process. Each active trace stream shall have a mapping that incorporates all the trace event type identifiers predefined by the trace system plus all the mappings of trace event names to trace event type identifiers of the processes that are being traced into that trace stream. These mappings are defined from the instrumented application by calling the *posix_trace_eventid_open*() function and from the controller process by calling the *posix_trace_trid_eventid_open*() function. For a pre-recorded trace stream, the list of trace event types is obtained from the pre-recorded trace log.

The *st_ctime* and *st_mtime* fields of a file associated with an active trace stream shall be marked for update every time any of the tracing operations modifies that file.

The *st_atime* field of a file associated with a trace stream shall be marked for update every time any of the tracing operations causes data to be read from that file.

Results are undefined if the application performs any operation on a file descriptor associated with an active or pre-recorded trace stream until *posix_trace_shutdown*() or *posix_trace_close*() is called for that trace stream.

The main purpose of this option is to define a complete set of functions and concepts that allow a conforming application to be traced from creation to termination, whatever its realtime constraints and properties.

2.11.1 Tracing Data Definitions

2.11.1.1 Structures

The **<trace.h>** header shall define the *posix_trace_status_info* and *posix_trace_event_info* structures described below. Implementations may add extensions to these structures.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

posix_trace_status_info Structure

To facilitate control of a trace stream, information about the current state of an active trace stream can be obtained dynamically. This structure is returned by a call to the *posix_trace_get_status*( ) function.

The **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following members:

Member Type	Member Name	Description
int	*posix_stream_status*	The operating mode of the trace stream.
int	*posix_stream_full_status*	The full status of the trace stream.
int	*posix_stream_overrun_status*	Indicates whether trace events were lost in the trace stream.

If the Trace Log option is supported in addition to the Trace option, the **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following additional members:

Member Type	Member Name	Description
int	*posix_stream_flush_status*	Indicates whether a flush is in progress.
int	*posix_stream_flush_error*	Indicates whether any error occurred during the last flush operation.
int	*posix_log_overrun_status*	Indicates whether trace events were lost in the trace log.
int	*posix_log_full_status*	The full status of the trace log.

The *posix_stream_status* member indicates the operating mode of the trace stream and shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_RUNNING
: Tracing is in progress; that is, the trace stream is accepting trace events.

POSIX_TRACE_SUSPENDED
: The trace stream is not accepting trace events. The tracing operation has not yet started or has stopped, either following a *posix_trace_stop*( ) function call or because the trace resources are exhausted.

The *posix_stream_full_status* member indicates the full status of the trace stream, and it shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_FULL
: The space in the trace stream for trace events is exhausted.

POSIX_TRACE_NOT_FULL
: There is still space available in the trace stream.

The combination of the *posix_stream_status* and *posix_stream_full_status* members also indicates the actual status of the stream. The status shall be interpreted as follows:

POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL
: This status combination indicates that tracing is in progress, and there is space available for recording more trace events.

POSIX_TRACE_RUNNING and POSIX_TRACE_FULL
: This status combination indicates that tracing is in progress and that the trace stream is full of trace events. This status combination cannot occur unless the *stream-full-policy* is set to

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving time window of prior trace events, and some older trace events may have been overwritten and thus lost.

POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL
This status combination indicates that tracing has not yet been started, has been stopped by the *posix_trace_stop*( ) function, or has been cleared by the *posix_trace_clear*( ) function.

POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL
This status combination indicates that tracing has been stopped by the implementation because the *stream-full-policy* attribute was POSIX_TRACE_UNTIL_FULL and trace resources were exhausted, or that the trace stream was stopped by the function *posix_trace_stop*( ) at a time when trace resources were exhausted.

The *posix_stream_overrun_status* member indicates whether trace events were lost in the trace stream, and shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_OVERRUN
At least one trace event was lost and thus was not recorded in the trace stream.

POSIX_TRACE_NO_OVERRUN
No trace events were lost.

When the corresponding trace stream is created, the *posix_stream_overrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN.

Whenever an overrun occurs, the *posix_stream_overrun_status* member shall be set to POSIX_TRACE_OVERRUN.

An overrun occurs when:

- The policy is POSIX_TRACE_LOOP and a recorded trace event is overwritten.
- The policy is POSIX_TRACE_UNTIL_FULL and the trace stream is full when a trace event is generated.
- If the Trace Log option is supported, the policy is POSIX_TRACE_FLUSH and at least one trace event is lost while flushing the trace stream to the trace log.

The *posix_stream_overrun_status* member is reset to zero after its value is read.

If the Trace Log option is supported in addition to the Trace option, the *posix_stream_flush_status*, *posix_stream_flush_error*, *posix_log_overrun_status*, and *posix_log_full_status* members are defined as follows; otherwise, they are undefined.

The *posix_stream_flush_status* member indicates whether a flush operation is being performed and shall have one of the following values defined by manifest constants in the header **<trace.h>**:

POSIX_TRACE_FLUSHING
The trace stream is currently being flushed to the trace log.

POSIX_TRACE_NOT_FLUSHING
No flush operation is in progress.

The *posix_stream_flush_status* member shall be set to POSIX_TRACE_FLUSHING if a flush operation is in progress either due to a call to the *posix_trace_flush*( ) function (explicit or caused by a trace stream shutdown operation) or because the trace stream has become full with the *stream-full-policy* attribute set to POSIX_TRACE_FLUSH. The *posix_stream_flush_status* member shall be set to POSIX_TRACE_NOT_FLUSHING if no flush operation is in progress.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *posix_stream_flush_error* member shall be set to zero if no error occurred during flushing. If an error occurred during a previous flushing operation, the *posix_stream_flush_error* member shall be set to the value of the first error that occurred. If more than one error occurs while flushing, error values after the first shall be discarded. The *posix_stream_flush_error* member is reset to zero after its value is read.

The *posix_log_overrun_status* member indicates whether trace events were lost in the trace log, and shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_OVERRUN
: At least one trace event was lost.

POSIX_TRACE_NO_OVERRUN
: No trace events were lost.

When the corresponding trace stream is created, the *posix_log_overrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN. Whenever an overrun occurs, this status shall be set to POSIX_TRACE_OVERRUN. The *posix_log_overrun_status* member is reset to zero after its value is read.

The *posix_log_full_status* member indicates the full status of the trace log, and it shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_FULL
: The space in the trace log is exhausted.

POSIX_TRACE_NOT_FULL
: There is still space available in the trace log.

The *posix_log_full_status* member is only meaningful if the *log-full-policy* attribute is either POSIX_TRACE_UNTIL_FULL or POSIX_TRACE_LOOP.

For an active trace stream without log, that is created by the *posix_trace_create*( ) function, the *posix_log_overrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN and the *posix_log_full_status* member shall be set to POSIX_TRACE_NOT_FULL.

posix_trace_event_info Structure

The trace event structure **posix_trace_event_info** contains the information for one recorded trace event. This structure is returned by the set of functions *posix_trace_getnext_event*( ), *posix_trace_timedgetnext_event*( ), and *posix_trace_trygetnext_event*( ).

The **posix_trace_event_info** structure defined in **<trace.h>** shall contain at least the following members:

Member Type	Member Name	Description
trace_event_id_t	*posix_event_id*	Trace event type identification.
pid_t	*posix_pid*	Process ID of the process that generated the trace event.
void *	*posix_prog_address*	Address at which the trace point was invoked.
int	*posix_truncation_status*	Status about the truncation of the data associated with this trace event.
struct timespec	*posix_timestamp*	Time at which the trace event was generated.

In addition, if the Trace option and the Threads option are both supported, the **posix_trace_event_info** structure defined in **<trace.h>** shall contain the following additional member:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Member Type	Member Name	Description
pthread_t	*posix_thread_id*	Thread ID of the thread that generated the trace event.

The *posix_event_id* member represents the identification of the trace event type and its value is not directly defined by the user. This identification is returned by a call to one of the following functions: *posix_trace_trid_eventid_open*( ), *posix_trace_eventtypelist_getnext_id*( ), or *posix_trace_eventid_open*( ). The name of the trace event type can be obtained by calling *posix_trace_eventid_get_name*( ).

The *posix_pid* is the process identifier of the traced process which generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any process, the *posix_pid* member shall be set to zero.

For a user trace event, the *posix_prog_address* member is the process mapped address of the point at which the associated call to the *posix_trace_event*( ) function was made. For a system trace event, if the trace event is caused by a system service explicitly called by the application, the *posix_prog_address* member shall be the address of the process at the point where the call to that system service was made.

The *posix_truncation_status* member defines whether the data associated with a trace event has been truncated at the time the trace event was generated, or at the time the trace event was read from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event* argument from the *posix_trace_getnext_event*( ) function). The *posix_truncation_status* member shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_NOT_TRUNCATED
: All the traced data is available.

POSIX_TRACE_TRUNCATED_RECORD
: Data was truncated at the time the trace event was generated.

POSIX_TRACE_TRUNCATED_READ
: Data was truncated at the time the trace event was read from a trace stream or a trace log because the reader's buffer was too small. This truncation status overrides the POSIX_TRACE_TRUNCATED_RECORD status.

The *posix_timestamp* member shall be the time at which the trace event was generated. The clock used is implementation-defined, but the resolution of this clock can be retrieved by a call to the *posix_trace_attr_getclockres*( ) function.

If the Threads option is supported in addition to the Trace option:

- The *posix_thread_id* member is the identifier of the thread that generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any thread, the *posix_thread_id* member shall be set to zero.

Otherwise, this member is undefined.

2.11.1.2 Trace Stream Attributes

Trace streams have attributes that compose the **posix_trace_attr_t** trace stream attributes object. This object shall contain at least the following attributes:

- The *generation-version* attribute identifies the origin and version of the trace system.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- The *trace-name* attribute is a character string defined by the trace controller, and that identifies the trace stream.
- The *creation-time* attribute represents the time of the creation of the trace stream.
- The *clock-resolution* attribute defines the clock resolution of the clock used to generate timestamps.
- The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly reserved for the trace events.
- The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its value is POSIX_TRACE_LOOP, POSIX_TRACE_UNTIL_FULL, or POSIX_TRACE_FLUSH.
- The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

In addition, if the Trace option and the Trace Inherit option are both supported, the **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following attributes:

- The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing in its parent's process trace stream. It is either POSIX_TRACE_INHERITED or POSIX_TRACE_CLOSE_FOR_CHILD.

In addition, if the Trace option and the Trace Log option are both supported, the **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following attribute:

- If the file type corresponding to the trace log supports the POSIX_TRACE_LOOP or the POSIX_TRACE_UNTIL_FULL policies, the *log-max-size* attribute defines the maximum size in bytes of the trace log associated with an active trace stream. Other stream data—for example, trace attribute values—shall not be included in this size.
- The *log-full-policy* attribute defines the policy of a trace log associated with an active trace stream to be POSIX_TRACE_LOOP, POSIX_TRACE_UNTIL_FULL, or POSIX_TRACE_APPEND.

2.11.2 Trace Event Type Definitions

2.11.2.1 System Trace Event Type Definitions

The following system trace event types, defined in the **<trace.h>** header, track the invocation of the trace operations:

- POSIX_TRACE_START shall be associated with a trace start operation.
- POSIX_TRACE_STOP shall be associated with a trace stop operation.
- If the Trace Event Filter option is supported, POSIX_TRACE_FILTER shall be associated with a trace event type filter change operation.

The following system trace event types, defined in the **<trace.h>** header, report operational trace events:

- POSIX_TRACE_OVERFLOW shall mark the beginning of a trace overflow condition.
- POSIX_TRACE_RESUME shall mark the end of a trace overflow condition.
- If the Trace Log option is supported, POSIX_TRACE_FLUSH_START shall mark the beginning of a flush operation.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- If the Trace Log option is supported, POSIX_TRACE_FLUSH_STOP shall mark the end of a flush operation.
- If an implementation-defined trace error condition is reported, it shall be marked POSIX_TRACE_ERROR.

The interpretation of a trace stream or a trace log by a trace analyzer process relies on the information recorded for each trace event, and also on system trace events that indicate the invocation of trace control operations and trace system operational trace events.

The POSIX_TRACE_START and POSIX_TRACE_STOP trace events specify the time windows during which the trace stream is running.

- The POSIX_TRACE_STOP trace event with an associated data that is equal to zero indicates a call of the function *posix_trace_stop*( ).
- The POSIX_TRACE_STOP trace event with an associated data that is different from zero indicates an automatic stop of the trace stream (see *posix_trace_attr_getstreamfullpolicy*( )).

The POSIX_TRACE_FILTER trace event indicates that a trace event type filter value changed while the trace stream was running.

The POSIX_TRACE_ERROR serves to inform the analyzer process that an implementation-defined internal error of the trace system occurred.

The POSIX_TRACE_OVERFLOW trace event shall be reported with a timestamp equal to the timestamp of the first trace event overwritten. This is an indication that some generated trace events have been lost.

The POSIX_TRACE_RESUME trace event shall be reported with a timestamp equal to the timestamp of the first valid trace event reported after the overflow condition ends and shall be reported before this first valid trace event. This is an indication that the trace system is reliably recording trace events after an overflow condition.

Each of these trace event types shall be defined by a constant trace event name and a **trace_event_id_t** constant; trace event data is associated with some of these trace events.

If the Trace option is supported and the Trace Event Filter option and the Trace Log option are not supported, the following predefined system trace events in Table 2-3 shall be defined:

Table 2-3 Trace Option: System Trace Events

Event Name	Constant	Associated Data Type
posix_trace_error	POSIX_TRACE_ERROR	error **int**
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto **int**
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log option is not supported, the following predefined system trace events in Table 2-4 (on page 78) shall be defined:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 2-4 Trace and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter option is not supported, the following predefined system trace events in Table 2-5 shall be defined:

Table 2-5 Trace and Trace Log Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in Table 2-6 (on page 79) shall be defined:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 2-6 Trace, Trace Log, and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data Data Type
posix_trace_error	POSIX_TRACE_ERROR	error **int**
posix_trace_start	POSIX_TRACE_START	event_filter **trace_event_set_t**
posix_trace_stop	POSIX_TRACE_STOP	auto **int**
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter new_event_filter **trace_event_set_t**
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

2.11.2.2 User Trace Event Type Definitions

The user trace event POSIX_TRACE_UNNAMED_USEREVENT is defined in the **<trace.h>** header. If the limit of per-process user trace event names represented by {TRACE_USER_EVENT_MAX} has already been reached, this predefined user event shall be returned when the application tries to register more events than allowed. The data associated with this trace event is application-defined.

The following predefined user trace event in Table 2-7 shall be defined:

Table 2-7 Trace Option: User Trace Event

Event Name	Constant
posix_trace_unnamed_userevent	POSIX_TRACE_UNNAMED_USEREVENT

2.11.3 Trace Functions

The trace interface is built and structured to improve portability through use of trace data of opaque type. The object-oriented approach for the manipulation of trace attributes and trace event type identifiers requires definition of many constructor and selector functions which operate on these opaque types. Also, the trace interface must support several different tracing roles. To facilitate reading the trace interface, the trace functions are grouped into small functional sets supporting the three different roles:

- A trace controller process requires functions to set up and customize all the resources needed to run a trace stream, including:
  - — Attribute initialization and destruction (*posix_trace_attr_init*())
  - — Identification information manipulation (*posix_trace_attr_getgenversion*())
  - — Trace system behavior modification (*posix_trace_attr_getinherited*())
  - — Trace stream and trace log size set (*posix_trace_attr_getmaxusereventsize*())

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— Trace stream creation, flush, and shutdown (*posix_trace_create*( ))

— Trace stream and trace log clear (*posix_trace_clear*( ))

— Trace event type identifier manipulation (*posix_trace_trid_eventid_open*( ))

— Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id*( ))

— Trace event type set manipulation (*posix_trace_eventset_empty*( ))

— Trace event type filter set (*posix_trace_set_filter*( ))

— Trace stream start and stop (*posix_trace_start*( ))

— Trace stream information and status read (*posix_trace_get_attr*( ))

- A traced process requires functions to instrument trace points:

  — Trace event type identifiers definition and trace points insertion (*posix_trace_event*( ))

- A trace analyzer process requires functions to retrieve information from a trace stream and trace log:

  — Identification information read (*posix_trace_attr_getgenversion*( ))

  — Trace system behavior information read (*posix_trace_attr_getinherited*( ))

  — Trace stream and trace log size get (*posix_trace_attr_getmaxusereventsize*( ))

  — Trace event type identifier manipulation (*posix_trace_trid_eventid_open*( ))

  — Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id*( ))

  — Trace log open, rewind, and close (*posix_trace_open*( ))

  — Trace stream information and status read (*posix_trace_get_attr*( ))

  — Trace event read (*posix_trace_getnext_event*( ))

2.12 Data Types

All of the data types used by various functions are defined by the implementation. The following table describes some of these types. Other types referenced in the description of a function, not mentioned here, can be found in the appropriate header for that function.

Defined Type	Description
cc_t	Type used for terminal special characters.
clock_t	Integer or real-floating type used for processor times, as defined in the ISO C standard.
clockid_t	Used for clock ID type in some timer functions.
dev_t	Arithmetic type used for device numbers.
DIR	Type representing a directory stream.
div_t	Structure type returned by the *div*( ) function.
FILE	Structure containing information about a file.
glob_t	Structure type used in pathname pattern matching.
fpos_t	Type containing all information needed to specify uniquely every

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Defined Type	Description
	position within a file.
gid_t	Integer type used for group IDs.
iconv_t	Type used for conversion descriptors.
id_t	Integer type used as a general identifier; can be used to contain at least the largest of a **pid_t**, **uid_t**, or **gid_t**.
ino_t	Unsigned integer type used for file serial numbers.
key_t	Arithmetic type used for XSI interprocess communication.
ldiv_t	Structure type returned by the *ldiv*( ) function.
mode_t	Integer type used for file attributes.
mqd_t	Used for message queue descriptors.
nfds_t	Integer type used for the number of file descriptors.
nlink_t	Integer type used for link counts.
off_t	Signed integer type used for file sizes.
pid_t	Signed integer type used for process and process group IDs.
pthread_attr_t	Used to identify a thread attribute object.
pthread_cond_t	Used for condition variables.
pthread_condattr_t	Used to identify a condition attribute object.
pthread_key_t	Used for thread-specific data keys.
pthread_mutex_t	Used for mutexes.
pthread_mutexattr_t	Used to identify a mutex attribute object.
pthread_once_t	Used for dynamic package initialization.
pthread_rwlock_t	Used for read-write locks.
pthread_rwlockattr_t	Used for read-write lock attributes.
pthread_t	Used to identify a thread.
ptrdiff_t	Signed integer type of the result of subtracting two pointers.
regex_t	Structure type used in regular expression matching.
regmatch_t	Structure type used in regular expression matching.
rlim_t	Unsigned integer type used for limit values, to which objects of type **int** and **off_t** can be cast without loss of value.
sem_t	Type used in performing semaphore operations.
sig_atomic_t	Integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
sigset_t	Integer or structure type of an object used to represent sets of signals.
size_t	Unsigned integer type used for size of objects.
speed_t	Type used for terminal baud rates.
ssize_t	Signed integer type used for a count of bytes or an error indication.
suseconds_t	Signed integer type used for time in microseconds.
tcflag_t	Type used for terminal modes.
time_t	Integer or real-floating type used for time in seconds, as defined in the ISO C standard.
timer_t	Used for timer ID returned by the *timer_create*( ) function.
uid_t	Integer type used for user IDs.
useconds_t	Unsigned integer type used for time in microseconds.
va_list	Type used for traversing variable argument lists.
wchar_t	Integer type whose range of values can represent distinct codes for

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Defined Type	Description
	all members of the largest extended character set specified by the supported locales.
wctype_t	Scalar type which represents a character class descriptor.
wint_t	Integer type capable of storing any valid value of **wchar_t** or WEOF.
wordexp_t	Structure type used in word expansion.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3356 *Chapter 3*

System Interfaces

3357 This chapter describes the functions, macros, and external variables to support applications
3358 portability at the C-language source level.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

FD_CLR — macros for synchronous I/O multiplexing

SYNOPSIS

```
#include <sys/time.h>

FD_CLR(int fd, fd_set *fdset);
FD_ISSET(int fd, fd_set *fdset);
FD_SET(int fd, fd_set *fdset);
FD_ZERO(fd_set *fdset);
```

DESCRIPTION

Refer to *pselect*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

_Exit, _exit — terminate a process

SYNOPSIS

```
#include <stdlib.h>
void _Exit(int status);
#include <unistd.h>
void _exit(int status);
```

DESCRIPTION

Refer to *exit*( ).

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

_longjmp, _setjmp — non-local goto

SYNOPSIS

XSI

```
#include <setjmp.h>

void _longjmp(jmp_buf env, int val);
int _setjmp(jmp_buf env);
```

DESCRIPTION

The *_longjmp*() and *_setjmp*() functions shall be equivalent to *longjmp*() and *setjmp*(), respectively, with the additional restriction that *_longjmp*() and *_setjmp*() shall not manipulate the signal mask.

If *_longjmp*() is called even though *env* was never initialized by a call to *_setjmp*(), or when the last such call was in a function that has since returned, the results are undefined.

RETURN VALUE

Refer to *longjmp*() and *setjmp*().

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

If *_longjmp*() is executed and the environment in which *_setjmp*() was executed no longer exists, errors can occur. The conditions under which the environment of the *_setjmp*() no longer exists include exiting the function that contains the *_setjmp*() call, and exiting an inner block with temporary storage. This condition might not be detectable, in which case the *_longjmp*() occurs and, if the environment no longer exists, the contents of the temporary storage of an inner block are unpredictable. This condition might also cause unexpected process termination. If the function has returned, the results are undefined.

Passing *longjmp*() a pointer to a buffer not created by *setjmp*(), passing *_longjmp*() a pointer to a buffer not created by *_setjmp*(), passing *siglongjmp*() a pointer to a buffer not created by *sigsetjmp*(), or passing any of these three functions a buffer that has been modified by the user can cause all the problems listed above, and more.

The *_longjmp*() and *_setjmp*() functions are included to support programs written to historical system interfaces. New applications should use *siglongjmp*() and *sigsetjmp*() respectively.

RATIONALE

None.

FUTURE DIRECTIONS

The *_longjmp*() and *_setjmp*() functions may be marked LEGACY in a future version.

SEE ALSO

longjmp(), *setjmp*(), *siglongjmp*(), *sigsetjmp*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<setjmp.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 5

Moved from X/OPEN UNIX extension to BASE.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

_tolower — transliterate uppercase characters to lowercase

SYNOPSIS

XSI
```
#include <ctype.h>

int _tolower(int c);
```

DESCRIPTION

The *_tolower*( ) macro shall be equivalent to *tolower*(*c*) except that the application shall ensure that the argument *c* is an uppercase letter.

RETURN VALUE

Upon successful completion, *_tolower*( ) shall return the lowercase letter corresponding to the argument passed.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tolower( ), *isupper*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

_toupper — transliterate lowercase characters to uppercase

SYNOPSIS

XSI
```
#include <ctype.h>

int _toupper(int c);
```

DESCRIPTION

The *_toupper*( ) macro shall be equivalent to *toupper*( ) except that the application shall ensure that the argument *c* is a lowercase letter.

RETURN VALUE

Upon successful completion, *_toupper*( ) shall return the uppercase letter corresponding to the argument passed.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

islower( ), *toupper*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

SYNOPSIS

XSI
```
#include <stdlib.h>

long a64l(const char *s);
char *l64a(long value);
```

DESCRIPTION

These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by which 32-bit integers can be represented by up to six characters; each character represents a digit in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall be used for these operations.

The characters used to represent digits are `'.'` (dot) for 0, `'/'` for 1, `'0'` through `'9'` for [2,11], `'A'` through `'Z'` for [12,37], and `'a'` through `'z'` for [38,63].

The *a64l*( ) function shall take a pointer to a radix-64 representation, in which the first digit is the least significant, and return the corresponding **long** value. If the string pointed to by *s* contains more than six characters, *a64l*( ) shall use the first six. If the first six characters of the string contain a null terminator, *a64l*( ) shall use only characters preceding the null terminator. The *a64l*( ) function shall scan the character string from left to right with the least significant digit on the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than 32 bits, the resulting value is sign-extended. The behavior of *a64l*( ) is unspecified if *s* is a null pointer or the string pointed to by *s* was not generated by a previous call to *l64a*( ).

The *l64a*( ) function shall take a **long** argument and return a pointer to the corresponding radix-64 representation. The behavior of *l64a*( ) is unspecified if *value* is negative.

The value returned by *l64a*( ) may be a pointer into a static buffer. Subsequent calls to *l64a*( ) may overwrite the buffer.

The *l64a*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *a64l*( ) shall return the **long** value resulting from conversion of the input string. If a string pointed to by *s* is an empty string, *a64l*( ) shall return 0L.

The *l64a*( ) function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a*( ) shall return a pointer to an empty string.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

If the type **long** contains more than 32 bits, the result of *a64l*(*l64a*(*x*)) is *x* in the low-order 32 bits.

RATIONALE

This is not the same encoding as used by either encoding variant of the *uuencode* utility.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

strtoul( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, *uuencode*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

abort — generate an abnormal process abort

SYNOPSIS

```
#include <stdlib.h>

void abort(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *abort*( ) function shall cause abnormal process termination to occur, unless the signal SIGABRT is being caught and the signal handler does not return.

CX The abnormal termination processing shall include the default actions defined for SIGABRT and may include an attempt to effect *fclose*( ) on all open streams.

The SIGABRT signal shall be sent to the calling process as if by means of *raise*( ) with the argument SIGABRT.

CX The status made available to *wait*( ) or *waitpid*( ) by *abort*( ) shall be that of a process terminated by the SIGABRT signal. The *abort*( ) function shall override blocking or ignoring the SIGABRT signal.

RETURN VALUE

The *abort*( ) function shall not return.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Catching the signal is intended to provide the application writer with a portable means to abort processing, free from possible interference from any implementation-defined functions.

RATIONALE

The ISO/IEC 9899:1999 standard requires the *abort*( ) function to be async-signal-safe. Since IEEE Std 1003.1-2001 defers to the ISO C standard, this required a change to the DESCRIPTION from ‘‘shall include the effect of *fclose*( )’’ to ‘‘may include an attempt to effect *fclose*( ).’’

The revised wording permits some backwards-compatibility and avoids a potential deadlock situation.

The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded paragraph from the DESCRIPTION:

‘‘On XSI-conformant systems, in addition the abnormal termination processing shall include the effect of *fclose*( ) on message catalog descriptors.’’

There were several reasons to remove this paragraph:

- No special processing of open message catalogs needs to be performed prior to abnormal process termination.
- The main reason to specifically mention that *abort*( ) includes the effect of *fclose*( ) on open streams is to flush output queued on the stream. Message catalogs in this context are read-only and, therefore, do not need to be flushed.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The effect of *fclose*() on a message catalog descriptor is unspecified. Message catalog descriptors are allowed, but not required to be implemented using a file descriptor, but there is no mention in IEEE Std 1003.1-2001 of a message catalog descriptor using a standard I/O stream FILE object as would be expected by *fclose*().

FUTURE DIRECTIONS

None.

SEE ALSO

exit(), *kill*(), *raise*(), *signal*(), *wait*(), *waitpid*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

The Open Group Base Resolution bwg2002-003 is applied.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

abs — return an integer absolute value

SYNOPSIS

```
#include <stdlib.h>

int abs(int i);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *abs*( ) function shall compute the absolute value of its integer operand, *i*. If the result cannot be represented, the behavior is undefined.

RETURN VALUE

The *abs*( ) function shall return the absolute value of its integer operand.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

In two's-complement representation, the absolute value of the negative integer with largest magnitude {INT_MIN} might not be representable.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fabs( ), *labs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

accept — accept a new connection on a socket

SYNOPSIS

```
#include <sys/socket.h>

int accept(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

DESCRIPTION

The *accept*( ) function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.

The *accept*( ) function takes the following arguments:

socket Specifies a socket that was created with *socket*( ), has been bound to an address with *bind*( ), and has issued a successful call to *listen*( ).

address Either a null pointer, or a pointer to a **sockaddr** structure where the address of the connecting socket shall be returned.

address_len Points to a **socklen_t** structure which on input specifies the length of the supplied **sockaddr** structure, and on output specifies the length of the stored address.

If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in the object pointed to by *address_len*.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file descriptor for the socket, *accept*( ) shall block until a connection is present. If the *listen*( ) queue is empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket, *accept*( ) shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

RETURN VALUE

Upon successful completion, *accept*( ) shall return the non-negative file descriptor of the accepted socket. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *accept*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]
O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.

[EBADF] The *socket* argument is not a valid file descriptor.

[ECONNABORTED]
A connection has been aborted.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[EINTR] The *accept*( ) function was interrupted by a signal that was caught before a valid connection arrived.

[EINVAL] The *socket* is not accepting connections.

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENFILE] The maximum number of file descriptors in the system are already open.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The socket type of the specified socket does not support accepting connections.

The *accept*( ) function may fail if:

[ENOBUFS] No buffer space is available.

[ENOMEM] There was insufficient memory available to complete the operation.

XSR [EPROTO] A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.

EXAMPLES

None.

APPLICATION USAGE

When a connection is available, *select*( ) indicates that the file descriptor for the socket is ready for reading.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

bind( ), *connect*( ), *listen*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *accept*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

access — determine accessibility of a file

SYNOPSIS

```
#include <unistd.h>

int access(const char *path, int amode);
```

DESCRIPTION

The *access*( ) function shall check the file named by the pathname pointed to by the *path* argument for accessibility according to the bit pattern contained in *amode*, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID.

The value of *amode* is either the bitwise-inclusive OR of the access permissions to be checked (R_OK, W_OK, X_OK) or the existence test (F_OK).

If any access permissions are checked, each shall be checked individually, as described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 3, Definitions. If the process has appropriate privileges, an implementation may indicate success for X_OK even if none of the execute file permission bits are set.

RETURN VALUE

If the requested access is permitted, *access*( ) succeeds and shall return 0; otherwise, –1 shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *access*( ) function shall fail if:

[EACCES] Permission bits of the file mode do not permit the requested access, or search permission is denied on a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

[EROFS] Write access is requested for a file on a read-only file system.

The *access*( ) function may fail if:

[EINVAL] The value of the *amode* argument is invalid.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

[ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being executed.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Testing for the Existence of a File

The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
#include <unistd.h>
...
int result;
const char *filename = "/tmp/myfile";

result = access (filename, F_OK);
```

APPLICATION USAGE

Additional values of *amode* other than the set defined in the description may be valid; for example, if a system has extended access controls.

RATIONALE

In early proposals, some inadequacies in the *access*( ) function led to the creation of an *eaccess*( ) function because:

1. Historical implementations of *access*( ) do not test file access correctly when the process' real user ID is superuser. In particular, they always return zero when testing execute permissions without regard to whether the file is executable.
2. The superuser has complete access to all files on a system. As a consequence, programs started by the superuser and switched to the effective user ID with lesser privileges cannot use *access*( ) to test their file access permissions.

However, the historical model of *eaccess*( ) does not resolve problem (1), so this volume of IEEE Std 1003.1-2001 now allows *access*( ) to behave in the desired way because several implementations have corrected the problem. It was also argued that problem (2) is more easily solved by using *open*( ), *chdir*( ), or one of the *exec* functions as appropriate and responding to the error, rather than creating a new function that would not be as reliable. Therefore, *eaccess*( ) is not included in this volume of IEEE Std 1003.1-2001.

The sentence concerning appropriate privileges and execute permission bits reflects the two possibilities implemented by historical implementations when checking superuser access for X_OK.

New implementations are discouraged from returning X_OK unless at least one execution permission bit is set.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *stat*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

acos, acosf, acosl — arc cosine functions

SYNOPSIS

```
#include <math.h>

double acos(double x);
float acosf(float x);
long double acosl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the principal value of the arc cosine of their argument *x*. The value of *x* should be in the range [–1,1].

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc cosine of *x*, in the range [0,π] radians.

MX For finite values of *x* not in the range [–1,1], a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is +1, +0 shall be returned.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is finite and is not in the range [–1,1], or is ±Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

cos( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *acosf*( ) and *acosl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

acosh, acoshf, acoshl — inverse hyperbolic cosine functions

SYNOPSIS

```
#include <math.h>

double acosh(double x);
float acoshf(float x);
long double acoshl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic cosine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic cosine of their argument.

MX For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is +1, +0 shall be returned.

If *x* is +Inf, +Inf shall be returned.

If *x* is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is finite and less than +1.0, or is –Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

cosh( ), *feclearexcept*( ), *fetestexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *acosh*( ) function is no longer marked as an extension.

The *acoshf*( ) and *acoshl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3891 **NAME**

3892 acosl — arc cosine functions

3893 **SYNOPSIS**

3894 `#include <math.h>`

3895 `long double acosl(long double x);`

3896 **DESCRIPTION**

3897 Refer to *acos*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

aio_cancel — cancel an asynchronous I/O request (**REALTIME**)

SYNOPSIS

AIO

```
#include <aio.h>

int aio_cancel(int fildes, struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_cancel*( ) function shall attempt to cancel one or more asynchronous I/O requests currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

Normal asynchronous notification shall occur for asynchronous I/O operations that are successfully canceled. If there are requests that cannot be canceled, then the normal asynchronous completion process shall take place for those requests when they are completed.

For requested operations that are successfully canceled, the associated error status shall be set to [ECANCELED] and the return status shall be –1. For requested operations that are not successfully canceled, the *aiocbp* shall not be modified by *aio_cancel*( ).

If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which the asynchronous operation was initiated, unspecified results occur.

Which operations are cancelable is implementation-defined.

RETURN VALUE

The *aio_cancel*( ) function shall return the value AIO_CANCELED to the calling process if the requested operation(s) were canceled. The value AIO_NOTCANCELED shall be returned if at least one of the requested operation(s) cannot be canceled because it is in progress. In this case, the state of the other operations, if any, referenced in the call to *aio_cancel*( ) is not indicated by the return value of *aio_cancel*( ). The application may determine the state of affairs for these operations by using *aio_error*( ). The value AIO_ALLDONE is returned if all of the operations have already completed. Otherwise, the function shall return –1 and set *errno* to indicate the error.

ERRORS

The *aio_cancel*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

EXAMPLES

None.

APPLICATION USAGE

The *aio_cancel*( ) function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3939 **SEE ALSO**

3940 *aio_read*( ), *aio_write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

3941 **CHANGE HISTORY**

3942 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

3943 **Issue 6**

3944 The [ENOSYS] error condition has been removed as stubs need not be provided if an
3945 implementation does not support the Asynchronous Input and Output option.

3946 The APPLICATION USAGE section is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

aio_error — retrieve errors status for an asynchronous I/O operation (**REALTIME**)

SYNOPSIS

AIO
```
#include <aio.h>

int aio_error(const struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_error*( ) function shall return the error status associated with the **aiocb** structure referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the SIO *errno* value that would be set by the corresponding *read*( ), *write*( ), *fdatasync*( ), or *fsync*( ) operation. If the operation has not yet completed, then the error status shall be equal to [EINPROGRESS].

RETURN VALUE

If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the asynchronous operation has completed unsuccessfully, then the error status, as described for SIO *read*( ), *write*( ), *fdatasync*( ), and *fsync*( ), shall be returned. If the asynchronous I/O operation has not yet completed, then [EINPROGRESS] shall be returned.

ERRORS

The *aio_error*( ) function may fail if:

[EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose return status has not yet been retrieved.

EXAMPLES

None.

APPLICATION USAGE

The *aio_error*( ) function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel( ), *aio_fsync*( ), *aio_read*( ), *aio_return*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ), *lseek*( ), *read*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

aio_fsync — asynchronous file synchronization (**REALTIME**)

SYNOPSIS

AIO

```
#include <aio.h>

int aio_fsync(int op, struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_fsync*( ) function shall asynchronously force all I/O operations associated with the file indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*( ) to the synchronized I/O completion state. The function call shall return when the synchronization request has been initiated or queued to the file or device (even when the data cannot be synchronized immediately).

If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to *fdatasync*( ); that is, as defined for synchronized I/O data integrity completion. If *op* is O_SYNC, all currently queued I/O operations shall be completed as if by a call to *fsync*( ); that is, as defined for synchronized I/O file integrity completion. If the *aio_fsync*( ) function fails, or if the operation queued by *aio_fsync*( ) fails, then, as for *fsync*( ) and *fdatasync*( ), outstanding I/O operations are not guaranteed to have been completed.

If *aio_fsync*( ) succeeds, then it is only the I/O that was queued at the time of the call to *aio_fsync*( ) that is guaranteed to be forced to the relevant completion state. The completion of subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized fashion.

The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used as an argument to *aio_error*( ) and *aio_return*( ) in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. When the request is queued, the error status for the operation is [EINPROGRESS]. When all data has been successfully transferred, the error status shall be reset to reflect the success or failure of the operation. If the operation does not complete successfully, the error status for the operation shall be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to occur as specified in Section 2.4.1 (on page 28) when all operations have achieved synchronized I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

If the *aio_fsync*( ) function fails or *aiocbp* indicates an error condition, data is not guaranteed to have been successfully transferred.

RETURN VALUE

The *aio_fsync*( ) function shall return the value 0 to the calling process if the I/O operation is successfully queued; otherwise, the function shall return the value –1 and set *errno* to indicate the error.

ERRORS

The *aio_fsync*( ) function shall fail if:

[EAGAIN] The requested asynchronous operation was not queued due to temporary resource limitations.

[EBADF] The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument is not a valid file descriptor open for writing.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EINVAL] This implementation does not support synchronized I/O for this file.

[EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified.

In the event that any of the queued I/O operations fail, *aio_fsync*() shall return the error condition defined for *read*() and *write*(). The error is returned in the error status for the asynchronous *fsync*() operation, which can be retrieved using *aio_error*().

EXAMPLES

None.

APPLICATION USAGE

The *aio_fsync*() function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fcntl(), *fdatasync*(), *fsync*(), *open*(), *read*(), *write*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

aio_read — asynchronous read from a file (**REALTIME**)

SYNOPSIS

AIO
```
#include <aio.h>

int aio_read(struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_read*( ) function shall read *aiocbp->aio_nbytes* from the file associated with *aiocbp->aio_fildes* into the buffer pointed to by *aiocbp->aio_buf*. The function call shall return when the read request has been initiated or queued to the file or device (even when the data cannot be delivered immediately).

PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio*.

The *aiocbp* value may be used as an argument to *aio_error*( ) and *aio_return*( ) in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. If an error condition is encountered during queuing, the function call shall return without having initiated or queued the request. The requested operation takes place at the absolute position in the file as given by *aio_offset*, as if *lseek*( ) were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_read*( ).

The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion.

For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*.

RETURN VALUE

The *aio_read*( ) function shall return the value zero to the calling process if the I/O operation is successfully queued; otherwise, the function shall return the value –1 and set *errno* to indicate the error.

ERRORS

The *aio_read*( ) function shall fail if:

[EAGAIN] The requested asynchronous I/O operation was not queued due to system resource limitations.

Each of the following conditions may be detected synchronously at the time of the call to *aio_read*( ), or asynchronously. If any of the conditions below are detected synchronously, the *aio_read*( ) function shall return –1 and set *errno* to the corresponding value. If any of the conditions below are detected asynchronously, the return status of the asynchronous operation

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

is set to –1, and the error status of the asynchronous operation is set to the corresponding value.

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for reading.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid, *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value.

In the case that the *aio_read*( ) successfully queues the I/O operation but the operation is subsequently canceled or encounters an error, the return status of the asynchronous operation is one of the values normally returned by the *read*( ) function call. In addition, the error status of the asynchronous operation is set to one of the error statuses normally set by the *read*( ) function call, or one of the following values:

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for reading.

[ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel*( ) request.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid.

The following condition may be detected synchronously or asynchronously:

[EOVERFLOW] The file is a regular file, *aiobcp->aio_nbytes* is greater than 0, and the starting offset in *aiobcp->aio_offset* is before the end-of-file and is at or beyond the offset maximum in the open file description associated with *aiocbp->aio_fildes*.

EXAMPLES

None.

APPLICATION USAGE

The *aio_read*( ) function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel( ), *aio_error*( ), *lio_listio*( ), *aio_return*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lseek*( ), *read*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

aio_return — retrieve return status of an asynchronous I/O operation (**REALTIME**)

SYNOPSIS

AIO

```
#include <aio.h>

ssize_t aio_return(struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_return*( ) function shall return the return status associated with the **aiocb** structure referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the value that would be returned by the corresponding *read*( ), *write*( ), or *fsync*( ) function call. If the error status for the operation is equal to [EINPROGRESS], then the return status for the operation is undefined. The *aio_return*( ) function may be called exactly once to retrieve the return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in a call to *aio_return*( ) or *aio_error*( ), an error may be returned. When the **aiocb** structure referred to by *aiocbp* is used to submit another asynchronous operation, then *aio_return*( ) may be successfully used to retrieve the return status of that operation.

RETURN VALUE

If the asynchronous I/O operation has completed, then the return status, as described for *read*( ), *write*( ), and *fsync*( ), shall be returned. If the asynchronous I/O operation has not yet completed, the results of *aio_return*( ) are undefined.

ERRORS

The *aio_return*( ) function may fail if:

[EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose return status has not yet been retrieved.

EXAMPLES

None.

APPLICATION USAGE

The *aio_return*( ) function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel( ), *aio_error*( ), *aio_fsync*( ), *aio_read*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ), *lseek*( ), *read*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

The [EINVAL] error condition is updated as a ‘‘may fail’’. This is for consistency with the DESCRIPTION.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

aio_suspend — wait for an asynchronous I/O request (**REALTIME**)

SYNOPSIS

AIO

```
#include <aio.h>

int aio_suspend(const struct aiocb * const list[], int nent,
    const struct timespec *timeout);
```

DESCRIPTION

The *aio_suspend*( ) function shall suspend the calling thread until at least one of the asynchronous I/O operations referenced by the *list* argument has completed, until a signal interrupts the function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is, the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the function shall return without suspending the calling thread. The *list* argument is an array of pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of elements in the array. Each **aiocb** structure pointed to has been used in initiating an asynchronous I/O request via *aio_read*( ), *aio_write*( ), or *lio_listio*( ). This array may contain NULL pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have not been used in submitting asynchronous I/O, the effect is undefined.

If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of the I/O operations referenced by *list* are completed, then *aio_suspend*( ) shall return with an error. MON If the Monotonic Clock option is supported, the clock that shall be used to measure this time interval shall be the CLOCK_MONOTONIC clock.

RETURN VALUE

If the *aio_suspend*( ) function returns after one or more asynchronous I/O operations have completed, the function shall return zero. Otherwise, the function shall return a value of –1 and set *errno* to indicate the error.

The application may determine which asynchronous I/O completed by scanning the associated error and return status using *aio_error*( ) and *aio_return*( ), respectively.

ERRORS

The *aio_suspend*( ) function shall fail if:

[EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the time interval indicated by *timeout*.

[EINTR] A signal interrupted the *aio_suspend*( ) function. Note that, since each asynchronous I/O operation may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited.

EXAMPLES

None.

APPLICATION USAGE

The *aio_suspend*( ) function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4230 **FUTURE DIRECTIONS**

4231 None.

4232 **SEE ALSO**

4233 *aio_read*( ), *aio_write*( ), *lio_listio*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

4234 **CHANGE HISTORY**

4235 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4236 **Issue 6**

4237 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4238 implementation does not support the Asynchronous Input and Output option.

4239 The APPLICATION USAGE section is added.

4240 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
4241 CLOCK_MONOTONIC clock, if supported, is used.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

aio_write — asynchronous write to a file (**REALTIME**)

SYNOPSIS

AIO
```
#include <aio.h>

int aio_write(struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_write*( ) function shall write *aiocbp->aio_nbytes* to the file associated with *aiocbp->aio_fildes* from the buffer pointed to by *aiocbp->aio_buf*. The function shall return when the write request has been initiated or, at a minimum, queued to the file or device.

PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio*.

The *aiocbp* argument may be used as an argument to *aio_error*( ) and *aio_return*( ) in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding.

The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take place at the absolute position in the file as given by *aio_offset*, as if *lseek*( ) were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET. If O_APPEND is set for the file descriptor, write operations append to the file in the same order as the calls were made. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_write*( ).

Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion, and synchronized I/O file integrity completion.

For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*.

RETURN VALUE

The *aio_write*( ) function shall return the value zero to the calling process if the I/O operation is successfully queued; otherwise, the function shall return –1 and set *errno* to indicate the error.

ERRORS

The *aio_write*( ) function shall fail if:

[EAGAIN] The requested asynchronous I/O operation was not queued due to system resource limitations.

Each of the following conditions may be detected synchronously at the time of the call to *aio_write*( ), or asynchronously. If any of the conditions below are detected synchronously, the *aio_write*( ) function shall return –1 and set *errno* to the corresponding value. If any of the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

conditions below are detected asynchronously, the return status of the asynchronous operation shall be set to –1, and the error status of the asynchronous operation is set to the corresponding value.

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid, *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value.

In the case that the *aio_write*( ) successfully queues the I/O operation, the return status of the asynchronous operation shall be one of the values normally returned by the *write*( ) function call. If the operation is successfully queued but is subsequently canceled or encounters an error, the error status for the asynchronous operation contains one of the values normally set by the *write*( ) function call, or one of the following:

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid.

[ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel*( ) request.

The following condition may be detected synchronously or asynchronously:

[EFBIG] The file is a regular file, *aiobcp->aio_nbytes* is greater than 0, and the starting offset in *aiobcp->aio_offset* is at or beyond the offset maximum in the open file description associated with *aiocbp->aio_fildes*.

EXAMPLES

None.

APPLICATION USAGE

The *aio_write*( ) function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel( ), *aio_error*( ), *aio_read*( ), *aio_return*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ), *lseek*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4329 *aiocbp->aio_fildes*.

4330 • The [EFBIG] error is added as part of the large file support extensions.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

alarm — schedule an alarm signal

SYNOPSIS

```
#include <unistd.h>

unsigned alarm(unsigned seconds);
```

DESCRIPTION

The *alarm*( ) function shall cause the system to generate a SIGALRM signal for the process after the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays may prevent the process from handling the signal as soon as it is generated.

If *seconds* is 0, a pending alarm request, if any, is canceled.

Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner. If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time at which the SIGALRM signal is generated.

XSI Interactions between *alarm*( ) and any of *setitimer*( ), *ualarm*( ), or *usleep*( ) are unspecified.

RETURN VALUE

If there is a previous *alarm*( ) request with time remaining, *alarm*( ) shall return a non-zero value that is the number of seconds until the previous request would have generated a SIGALRM signal. Otherwise, *alarm*( ) shall return 0.

ERRORS

The *alarm*( ) function is always successful, and no return value is reserved to indicate an error.

EXAMPLES

None.

APPLICATION USAGE

The *fork*( ) function clears pending alarms in the child process. A new process image created by one of the *exec* functions inherits the time left to an alarm signal in the old process' image.

Application writers should note that the type of the argument *seconds* and the return value of *alarm*( ) is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the ISO C standard sets as 65 535, and any application passing a larger value is restricting its portability. A different type was considered, but historical implementations, including those with a 16-bit **int** type, consistently use either **unsigned** or **int**.

Application writers should be aware of possible interactions when the same process uses both the *alarm*( ) and *sleep*( ) functions.

RATIONALE

Many historical implementations (including Version 7 and System V) allow an alarm to occur up to a second early. Other implementations allow alarms up to half a second or one clock tick early or do not allow them to occur early at all. The latter is considered most appropriate, since it gives the most predictable behavior, especially since the signal can always be delayed for an indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument as the minimum amount of time they wish to have elapse before the signal.

The term ''realtime'' here and elsewhere (*sleep*( ), *times*( )) is intended to mean ''wall clock'' time as common English usage, and has nothing to do with ''realtime operating systems''. It is in contrast to *virtual time*, which could be misinterpreted if just *time* were used.

In some implementations, including 4.3 BSD, very large values of the *seconds* argument are silently rounded down to an implementation-defined maximum value. This maximum is large

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

enough (to the order of several months) that the effect is not noticeable.

There were two possible choices for alarm generation in multi-threaded applications: generation for the calling thread or generation for the process. The first option would not have been particularly useful since the alarm state is maintained on a per-process basis and the alarm that is established by the last invocation of *alarm*( ) is the only one that would be active.

Furthermore, allowing generation of an asynchronous signal for a thread would have introduced an exception to the overall signal model. This requires a compelling reason in order to be justified.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *exec*, *fork*( ), *getitimer*( ), *pause*( ), *sigaction*( ), *sleep*( ), *ualarm*( ), *usleep*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate that interactions with the *setitimer*( ), *ualarm*( ), and *usleep*( ) functions are unspecified.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

asctime, asctime_r — convert date and time to a string

SYNOPSIS

```
#include <time.h>

char *asctime(const struct tm *timeptr);
```

TSF
```
char *asctime_r(const struct tm *restrict tm, char *restrict buf);
```

DESCRIPTION

CX For *asctime*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *asctime*( ) function shall convert the broken-down time in the structure pointed to by *timeptr* into a string in the form:

```
Sun Sep 16 01:03:52 1973\n\0
```

using the equivalent of the following algorithm:

```
char *asctime(const struct tm *timeptr)
{
    static char wday_name[7][3] = {
        "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
    };
    static char mon_name[12][3] = {
        "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
    };
    static char result[26];

    sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
        wday_name[timeptr->tm_wday],
        mon_name[timeptr->tm_mon],
        timeptr->tm_mday, timeptr->tm_hour,
        timeptr->tm_min, timeptr->tm_sec,
        1900 + timeptr->tm_year);
    return result;
}
```

The **tm** structure is defined in the **<time.h>** header.

CX The *asctime*( ), *ctime*( ), *gmtime*( ), and *localtime*( ) functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The *asctime*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

TSF The *asctime_r*( ) function shall convert the broken-down time in the structure pointed to by *tm* into a string (of the same form as that returned by *asctime*( )) that is placed in the user-supplied buffer pointed to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RETURN VALUE

Upon successful completion, *asctime*( ) shall return a pointer to the string.

TSF Upon successful completion, *asctime_r*( ) shall return a pointer to a character string containing the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful, it shall return NULL.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Values for the broken-down time structure can be obtained by calling *gmtime*( ) or *localtime*( ). This function is included for compatibility with older implementations, and does not support localized date and time formats. Applications should use *strftime*( ) to achieve maximum portability.

The *asctime_r*( ) function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock( ), *ctime*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

The *asctime_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *asctime*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *asctime_r*( ) function is marked as part of the Thread-Safe Functions option.

Extensions beyond the ISO C standard are marked.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The DESCRIPTION of *asctime_r*( ) is updated to describe the format of the string returned.

The **restrict** keyword is added to the *asctime_r*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

asin, asinf, asinl — arc sine function

SYNOPSIS

```
#include <math.h>

double asin(double x);
float asinf(float x);
long double asinl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the principal value of the arc sine of their argument *x*. The value of *x* should be in the range [–1,1].

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc sine of *x*, in the range $[-\pi/2,\pi/2]$ radians.

MX For finite values of *x* not in the range [–1,1], a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is finite and is not in the range [–1,1], or is ±Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The value of *x* is subnormal.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *sin*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *asinf*( ) and *asinl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

asinh, asinhf, asinhl — inverse hyperbolic sine functions

SYNOPSIS

```
#include <math.h>

double asinh(double x);
float asinhf(float x);
long double asinhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic sine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic sine of their argument.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, or ±Inf, *x* shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions may fail if:

MX

Range Error	The value of *x* is subnormal.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *sinh*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *asinh*( ) function is no longer marked as an extension.

The *asinhf*( ) and *asinhl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

4594 **NAME**

4595 asinl — arc sine function

4596 **SYNOPSIS**

4597
```
#include <math.h>
```

4598
```
long double asinl(long double x);
```

4599 **DESCRIPTION**

4600 Refer to *asin*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

assert — insert program diagnostics

SYNOPSIS

```
#include <assert.h>

void assert(scalar expression);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *assert*( ) macro shall insert diagnostics into programs; it shall expand to a **void** expression. When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal to 0), *assert*( ) shall write information about the particular call that failed on *stderr* and shall call *abort*( ).

The information written about the call that failed shall include the text of the argument, the name of the source file, the source file line number, and the name of the enclosing function; the latter are, respectively, the values of the preprocessing macros __FILE__ and __LINE__ and of the identifier __func__.

Forcing a definition of the name NDEBUG, either from the compiler command line or with the preprocessor control statement **#define** NDEBUG ahead of the **#include <assert.h>** statement, shall stop assertions from being compiled into the program.

RETURN VALUE

The *assert*( ) macro shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abort( ), *stderr*, the Base Definitions volume of IEEE Std 1003.1-2001, **<assert.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The prototype for the *expression* argument to *assert*( ) is changed from **int** to **scalar** for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION of *assert*( ) is updated for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

atan, atanf, atanl — arc tangent function

SYNOPSIS

```
#include <math.h>

double atan(double x);
float atanf(float x);
long double atanl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of *x* in the range $[-\pi/2,\pi/2]$ radians.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, $\pm\pi/2$ shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions may fail if:

MX

Range Error	The value of *x* is subnormal.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *atanf*( ) and *atanl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

atan2, atan2f, atan2l — arc tangent functions

SYNOPSIS

```
#include <math.h>

double atan2(double y, double x);
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of y/x in the range $[-\pi,\pi]$ radians.

If y is ± 0 and x is < 0, $\pm\pi$ shall be returned.

If y is ± 0 and x is > 0, ± 0 shall be returned.

If y is < 0 and x is ± 0, $-\pi/2$ shall be returned.

If y is > 0 and x is ± 0, $\pi/2$ shall be returned.

If x is 0, a pole error shall not occur.

MX If either x or y is NaN, a NaN shall be returned.

If the result underflows, a range error may occur and y/x should be returned.

If y is ± 0 and x is -0, $\pm\pi$ shall be returned.

If y is ± 0 and x is $+0$, ± 0 shall be returned.

For finite values of $\pm y > 0$, if x is $-$Inf, $\pm\pi$ shall be returned.

For finite values of $\pm y > 0$, if x is $+$Inf, ± 0 shall be returned.

For finite values of x, if y is $\pm$Inf, $\pm\pi/2$ shall be returned.

If y is $\pm$Inf and x is $-$Inf, $\pm 3\pi/4$ shall be returned.

If y is $\pm$Inf and x is $+$Inf, $\pm\pi/4$ shall be returned.

If both arguments are 0, a domain error shall not occur.

ERRORS

These functions may fail if:

MX Range Error — The result underflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

(math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *atan2f*( ) and *atan2l*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4762 **NAME**

4763 atanf — arc tangent function

4764 **SYNOPSIS**

4765 `#include <math.h>`

4766 `float atanf(float x);`

4767 **DESCRIPTION**

4768 Refer to *atan*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

atanh, atanhf, atanhl — inverse hyperbolic tangent functions

SYNOPSIS

```
#include <math.h>

double atanh(double x);
float atanhf(float x);
long double atanhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic tangent of their argument.

If *x* is ±1, a pole error shall occur, and *atanh*( ), *atanhf*( ), and *atanhl*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the correct value of the function.

MX For finite $|x|>1$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The *x* argument is finite and not in the range [–1,1], or is ±Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The *x* argument is ±1.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

-`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

These functions may fail if:

MX

Range Error	The value of *x* is subnormal.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *tanh*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *atanh*( ) function is no longer marked as an extension.

The *atanhf*( ) and *atanhl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4840 **NAME**

4841 atanl — arc tangent function

4842 **SYNOPSIS**

4843 `#include <math.h>`

4844 `long double atanl(long double x);`

4845 **DESCRIPTION**

4846 Refer to *atan*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

atexit — register a function to run at process termination

SYNOPSIS

```
#include <stdlib.h>

int atexit(void (*func)(void));
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *atexit*() function shall register the function pointed to by *func*, to be called without arguments at normal program termination. At normal program termination, all functions registered by the *atexit*() function shall be called, in the reverse order of their registration, except that a function is called after any previously registered functions that had already been called at the time it was registered. Normal termination occurs either by a call to *exit*() or a return from *main*().

At least 32 functions can be registered with *atexit*().

CX After a successful call to any of the *exec* functions, any functions previously registered by *atexit*() shall no longer be registered.

RETURN VALUE

Upon successful completion, *atexit*() shall return 0; otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The functions registered by a call to *atexit*() must return to ensure that all registered functions are called.

The application should call *sysconf*() to obtain the value of {ATEXIT_MAX}, the number of functions that can be registered. There is no way for an application to tell how many functions have already been registered with *atexit*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exit(), *sysconf*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4888 **NAME**

4889 atof — convert a string to a double-precision number

4890 **SYNOPSIS**

4891 `#include <stdlib.h>`

4892 `double atof(const char *str);`

4893 **DESCRIPTION**

4894 CX The functionality described on this reference page is aligned with the ISO C standard. Any
4895 conflict between the requirements described here and the ISO C standard is unintentional. This
4896 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4897 The call *atof*(*str*) shall be equivalent to:

4898 `strtod(str,(char **)NULL),`

4899 except that the handling of errors may differ. If the value cannot be represented, the behavior is
4900 undefined.

4901 **RETURN VALUE**

4902 The *atof*( ) function shall return the converted value if the value can be represented.

4903 **ERRORS**

4904 No errors are defined.

4905 **EXAMPLES**

4906 None.

4907 **APPLICATION USAGE**

4908 The *atof*( ) function is subsumed by *strtod*( ) but is retained because it is used extensively in
4909 existing code. If the number is not known to be in range, *strtod*( ) should be used because *atof*( ) is
4910 not required to perform any error checking.

4911 **RATIONALE**

4912 None.

4913 **FUTURE DIRECTIONS**

4914 None.

4915 **SEE ALSO**

4916 *strtod*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

4917 **CHANGE HISTORY**

4918 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

atoi — convert a string to an integer

SYNOPSIS

```
#include <stdlib.h>

int atoi(const char *str);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The call *atoi*(*str*) shall be equivalent to:

```
(int) strtol(str, (char **)NULL, 10)
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

The *atoi*( ) function shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

EXAMPLES

Converting an Argument

The following example checks for proper usage of the program. If there is an argument and the decimal conversion of this argument (obtained using *atoi*( )) is greater than 0, then the program has a valid number of minutes to wait for an event.

```
#include <stdlib.h>
#include <stdio.h>
...
int minutes_to_event;
...
if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
   fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
}
...
```

APPLICATION USAGE

The *atoi*( ) function is subsumed by *strtol*( ) but is retained because it is used extensively in existing code. If the number is not known to be in range, *strtol*( ) should be used because *atoi*( ) is not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4960 **CHANGE HISTORY**

4961 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

atol, atoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>

long atol(const char *str);
long long atoll(const char *nptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The call *atol*(*str*) shall be equivalent to:

```
strtol(str, (char **)NULL, 10)
```

The call *atoll*(*str*) shall be equivalent to:

```
strtoll(nptr, (char **)NULL, 10)
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

These functions shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *atol*( ) function is subsumed by *strtol*( ) but is retained because it is used extensively in existing code. If the number is not known to be in range, *strtol*( ) should be used because *atol*( ) is not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *atoll*( ) function is added for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

basename — return the last component of a pathname

SYNOPSIS

XSI
```
#include <libgen.h>

char *basename(char *path);
```

DESCRIPTION

The *basename*( ) function shall take the pathname pointed to by *path* and return a pointer to the final component of the pathname, deleting any trailing '/' characters.

If the string consists entirely of the '/' character, *basename*( ) shall return a pointer to the string "/". If the string is exactly "//", it is implementation-defined whether '/' or "//" is returned.

If *path* is a null pointer or points to an empty string, *basename*( ) shall return a pointer to the string ".".

The *basename*( ) function may modify the string pointed to by *path*, and may return a pointer to static storage that may then be overwritten by a subsequent call to *basename*( ).

The *basename*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *basename*( ) function shall return a pointer to the final component of *path*.

ERRORS

No errors are defined.

EXAMPLES

Using basename( )

The following program fragment returns a pointer to the value *lib*, which is the base name of **/usr/lib**.

```
#include <libgen.h>
...
char *name = "/usr/lib";
char *base;

base = basename(name);
...
```

Sample Input and Output Strings for basename( )

In the following table, the input string is the value pointed to by *path*, and the output string is the return value of the *basename*( ) function.

Input String	Output String
"/usr/lib"	"lib"
"/usr/"	"usr"
"/"	"/"
"///"	"/"
"//usr//lib//"	"lib"

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5039 **APPLICATION USAGE**

5040 None.

5041 **RATIONALE**

5042 None.

5043 **FUTURE DIRECTIONS**

5044 None.

5045 **SEE ALSO**

5046 *dirname*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<libgen.h>**, the Shell and
5047 Utilities volume of IEEE Std 1003.1-2001, *basename*

5048 **CHANGE HISTORY**

5049 First released in Issue 4, Version 2.

5050 **Issue 5**

5051 Moved from X/OPEN UNIX extension to BASE.

5052 Normative text previously in the APPLICATION USAGE section is moved to the
5053 DESCRIPTION.

5054 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5055 **Issue 6**

5056 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

bcmp — memory operations (**LEGACY**)

SYNOPSIS

XSI
```
#include <strings.h>

int bcmp(const void *s1, const void *s2, size_t n);
```

DESCRIPTION

The *bcmp*( ) function shall compare the first *n* bytes of the area pointed to by *s1* with the area pointed to by *s2*.

RETURN VALUE

The *bcmp*( ) function shall return 0 if *s1* and *s2* are identical; otherwise, it shall return non-zero. Both areas are assumed to be *n* bytes long. If the value of *n* is 0, *bcmp*( ) shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *memcmp*( ) function is preferred over this function.

For maximum portability, it is recommended to replace the function call to *bcmp*( ) as follows:

```
#define bcmp(b1,b2,len) memcmp((b1), (b2), (size_t)(len))
```

RATIONALE

None.

FUTURE DIRECTIONS

This function may be withdrawn in a future version.

SEE ALSO

memcmp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

This function is marked LEGACY.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

bcopy — memory operations (**LEGACY**)

SYNOPSIS

XSI

```
#include <strings.h>

void bcopy(const void *s1, void *s2, size_t n);
```

DESCRIPTION

The *bcopy*( ) function shall copy *n* bytes from the area pointed to by *s1* to the area pointed to by *s2*.

The bytes are copied correctly even if the area pointed to by *s1* overlaps the area pointed to by *s2*.

RETURN VALUE

The *bcopy*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *memmove*( ) function is preferred over this function.

The following are approximately equivalent (note the order of the arguments):

```
bcopy(s1,s2,n) ~= memmove(s2,s1,n)
```

For maximum portability, it is recommended to replace the function call to *bcopy*( ) as follows:

```
#define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void) 0)
```

RATIONALE

None.

FUTURE DIRECTIONS

This function may be withdrawn in a future version.

SEE ALSO

memmove( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

This function is marked LEGACY.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

bind — bind a name to a socket

SYNOPSIS

```
#include <sys/socket.h>

int bind(int socket, const struct sockaddr *address,
    socklen_t address_len);
```

DESCRIPTION

The *bind*( ) function shall assign a local socket address *address* to a socket identified by descriptor *socket* that has no local socket address assigned. Sockets created with the *socket*( ) function are initially unnamed; they are identified only by their address family.

The *bind*( ) function takes the following arguments:

socket Specifies the file descriptor of the socket to be bound.

address Points to a **sockaddr** structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.

address_len Specifies the length of the **sockaddr** structure pointed to by the *address* argument.

The socket specified by *socket* may require the process to have appropriate privileges to use the *bind*( ) function.

RETURN VALUE

Upon successful completion, *bind*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *bind*( ) function shall fail if:

[EADDRINUSE] The specified address is already in use.

[EADDRNOTAVAIL] The specified address is not available from the local machine.

[EAFNOSUPPORT] The specified address is not a valid address for the address family of the specified socket.

[EBADF] The *socket* argument is not a valid file descriptor.

[EINVAL] The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The socket type of the specified socket does not support binding to an address.

If the address family of the socket is AF_UNIX, then *bind*( ) shall fail if:

[EACCES] A component of the path prefix denies search permission, or the requested name requires writing in a directory with a mode that denies write permission.

[EDESTADDRREQ] or [EISDIR] The *address* argument is a null pointer.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[EIO] An I/O error occurred.

[ELOOP] A loop exists in symbolic links encountered during resolution of the pathname in *address*.

[ENAMETOOLONG]
A component of a pathname exceeded {NAME_MAX} characters, or an entire pathname exceeded {PATH_MAX} characters.

[ENOENT] A component of the pathname does not name an existing file or the pathname is an empty string.

[ENOTDIR] A component of the path prefix of the pathname in *address* is not a directory.

[EROFS] The name would reside on a read-only file system.

The *bind*( ) function may fail if:

[EACCES] The specified address is protected and the current user does not have permission to bind to it.

[EINVAL] The *address_len* argument is not a valid length for the address family.

[EISCONN] The socket is already connected.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in *address*.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[ENOBUFS] Insufficient resources were available to complete the call.

EXAMPLES

None.

APPLICATION USAGE

An application program can retrieve the assigned socket name with the *getsockname*( ) function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

connect( ), *getsockname*( ), *listen*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

bsd_signal — simplified signal facilities

SYNOPSIS

OB XSI

```
#include <signal.h>

void (*bsd_signal(int sig, void (*func)(int)))(int);
```

DESCRIPTION

The *bsd_signal*( ) function provides a partially compatible interface for programs written to historical system interfaces (see APPLICATION USAGE).

The function call *bsd_signal*(*sig*, *func*) shall be equivalent to the following:

```
void (*bsd_signal(int sig, void (*func)(int)))(int)
{
    struct sigaction act, oact;

    act.sa_handler = func;
    act.sa_flags = SA_RESTART;
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask, sig);
    if (sigaction(sig, &act, &oact) == -1)
        return(SIG_ERR);
    return(oact.sa_handler);
}
```

The handler function should be declared:

```
void handler(int sig);
```

where *sig* is the signal number. The behavior is undefined if *func* is a function that takes more than one argument, or an argument of a different type.

RETURN VALUE

Upon successful completion, *bsd_signal*( ) shall return the previous action for *sig*. Otherwise, SIG_ERR shall be returned and *errno* shall be set to indicate the error.

ERRORS

Refer to *sigaction*( ).

EXAMPLES

None.

APPLICATION USAGE

This function is a direct replacement for the BSD *signal*( ) function for simple applications that are installing a single-argument signal handler function. If a BSD signal handler function is being installed that expects more than one argument, the application has to be modified to use *sigaction*( ). The *bsd_signal*( ) function differs from *signal*( ) in that the SA_RESTART flag is set and the SA_RESETHAND is clear when *bsd_signal*( ) is used. The state of these flags is not specified for *signal*( ).

It is recommended that new applications use the *sigaction*( ) function.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

sigaction( ), *sigaddset*( ), *sigemptyset*( ), *signal*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

This function is marked obsolescent.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

bsearch — binary search a sorted table

SYNOPSIS

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t nel,
    size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *bsearch*( ) function shall search an array of *nel* objects, the initial element of which is pointed to by *base*, for an element that matches the object pointed to by *key*. The size of each element in the array is specified by *width*. If the *nel* argument has the value zero, the comparison function pointed to by *compar* shall not be called and no match shall be found.

The comparison function pointed to by *compar* shall be called with two arguments that point to the *key* object and to an array element, in that order.

The application shall ensure that the comparison function pointed to by *compar* does not alter the contents of the array. The implementation may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

The implementation shall ensure that the first argument is always a pointer to the key.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, the same object shall always compare the same way with the key.

The application shall ensure that the function returns an integer less than, equal to, or greater than 0 if the *key* object is considered, respectively, to be less than, to match, or to be greater than the array element. The application shall ensure that the array consists of all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the *key* object, in that order.

RETURN VALUE

The *bsearch*( ) function shall return a pointer to a matching member of the array, or a null pointer if no match is found. If two or more members compare equal, which member is returned is unspecified.

ERRORS

No errors are defined.

EXAMPLES

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

The code fragment below reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TABSIZE    1000
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
5296        struct node {                   /* These are stored in the table. */
5297            char *string;
5298            int length;
5299        };
5300        struct node table[TABSIZE];     /* Table to be searched. */
5301            .
5302            .
5303            .
5304        {
5305            struct node *node_ptr, node;
5306            /* Routine to compare 2 nodes. */
5307            int node_compare(const void *, const void *);
5308            char str_space[20];   /* Space to read string into. */
5309            .
5310            .
5311            .
5312            node.string = str_space;
5313            while (scanf("%s", node.string) != EOF) {
5314                node_ptr = (struct node *)bsearch((void *)(&node),
5315                       (void *)table, TABSIZE,
5316                       sizeof(struct node), node_compare);
5317                if (node_ptr != NULL) {
5318                    (void)printf("string = %20s, length = %d\n",
5319                        node_ptr->string, node_ptr->length);
5320                } else {
5321                    (void)printf("not found: %s\n", node.string);
5322                }
5323            }
5324        }
5325        /*
5326            This routine compares two nodes based on an
5327            alphabetical ordering of the string field.
5328        */
5329        int
5330        node_compare(const void *node1, const void *node2)
5331        {
5332            return strcoll(((const struct node *)node1)->string,
5333                ((const struct node *)node2)->string);
5334        }
```

5335 **APPLICATION USAGE**

5336 The pointers to the key and the element at the base of the table should be of type pointer-to-
5337 element.

5338 The comparison function need not compare every byte, so arbitrary data may be contained in
5339 the elements in addition to the values being compared.

5340 In practice, the array is usually sorted according to the comparison function.

5341 **RATIONALE**

5342 The requirement that the second argument (hereafter referred to as *p*) to the comparison
5343 function is a pointer to an element of the array implies that for every call all of the following
5344 expressions are non-zero:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
        ((char *)p - (char *(base) % width == 0
        (char *)p >= (char *)base
        (char *)p < (char *)base + nel * width
```

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *lsearch*( ), *qsort*( ), *tsearch*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three paragraphs. The RATIONALE section is also updated. These changes are for alignment with the ISO C standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

btowc — single byte to wide character conversion

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t btowc(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *btowc*( ) function shall determine whether *c* constitutes a valid (one-byte) character in the initial shift state.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *btowc*( ) function shall return WEOF if *c* has the value EOF or if **(unsigned char)** *c* does not constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the wide-character representation of that character.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wctob( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

bzero — memory operations (**LEGACY**)

SYNOPSIS

XSI

```
#include <strings.h>

void bzero(void *s, size_t n);
```

DESCRIPTION

The *bzero*( ) function shall place *n* zero-valued bytes in the area pointed to by *s*.

RETURN VALUE

The *bzero*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *memset*( ) function is preferred over this function.

For maximum portability, it is recommended to replace the function call to *bzero*( ) as follows:

```
#define bzero(b,len) (memset((b), '\0', (len)), (void) 0)
```

RATIONALE

None.

FUTURE DIRECTIONS

This function may be withdrawn in a future version.

SEE ALSO

memset( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

This function is marked LEGACY.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

cabs, cabsf, cabsl — return a complex absolute value

SYNOPSIS

```
#include <complex.h>

double cabs(double complex z);
float cabsf(float complex z);
long double cabsl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex absolute value (also called norm, modulus, or magnitude) of *z*.

RETURN VALUE

These functions shall return the complex absolute value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

cacos, cacosf, cacosl — complex arc cosine functions

SYNOPSIS

```
#include <complex.h>

double complex cacos(double complex z);
float complex cacosf(float complex z);
long double complex cacosl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex arc cosine of *z*, with branch cuts outside the interval [–1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc cosine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [0, π] along the real axis.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ccos( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

SYNOPSIS

```
#include <complex.h>

double complex cacosh(double complex z);
float complex cacoshf(float complex z);
long double complex cacoshl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic cosine of *z*, with a branch cut at values less than 1 along the real axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip of non-negative values along the real axis and in the interval [$-i\pi$, $+i\pi$] along the imaginary axis.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ccosh( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5512 **NAME**

5513 cacosl — complex arc cosine functions

5514 **SYNOPSIS**

5515 `#include <complex.h>`

5516 `long double complex cacosl(long double complex z);`

5517 **DESCRIPTION**

5518 Refer to *cacos*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

calloc — a memory allocator

SYNOPSIS

```
#include <stdlib.h>

void *calloc(size_t nelem, size_t elsize);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *calloc*( ) function shall allocate unused space for an array of *nelem* elements each of whose size in bytes is *elsize*. The space shall be initialized to all bits 0.

The order and contiguity of storage allocated by successive calls to *calloc*( ) is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object or an array of such objects in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned shall point to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-defined: the value returned shall be either a null pointer or a unique pointer.

RETURN VALUE

Upon successful completion with both *nelem* and *elsize* non-zero, *calloc*( ) shall return a pointer to the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer value that can be successfully passed to *free*( ) shall be returned. Otherwise, it shall return a null pointer CX and set *errno* to indicate the error.

ERRORS

The *calloc*( ) function shall fail if:

CX [ENOMEM] Insufficient memory is available.

EXAMPLES

None.

APPLICATION USAGE

There is now no requirement for the implementation to support the inclusion of **<malloc.h>**.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *malloc*( ), *realloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient memory condition occurs.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`-,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

carg, cargf, cargl — complex argument functions

SYNOPSIS

```
#include <complex.h>

double carg(double complex z);
float cargf(float complex z);
long double cargl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the argument (also called phase angle) of *z*, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the value of the argument in the interval [–π, +π].

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cimag( ), *conj*( ), *cproj*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

casin, casinf, casinl — complex arc sine functions

SYNOPSIS

```
#include <complex.h>

double complex casin(double complex z);
float complex casinf(float complex z);
long double complex casinl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex arc sine of *z*, with branch cuts outside the interval [–1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc sine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [$-\pi/2$, $+\pi/2$] along the real axis.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

csin( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

casinh, casinhf, casinhl — complex arc hyperbolic sine functions

SYNOPSIS

```
#include <complex.h>

double complex casinh(double complex z);
float complex casinhf(float complex z);
long double complex casinhl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic sine of *z*, with branch cuts outside the interval [–*i*, +*i*] along the imaginary axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic sine value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the imaginary axis.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

csinh( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5654 **NAME**

5655 casinl — complex arc sine functions

5656 **SYNOPSIS**

5657
```
#include <complex.h>
```

5658
```
long double complex casinl(long double complex z);
```

5659 **DESCRIPTION**

5660 Refer to *casin*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

catan, catanf, catanl — complex arc tangent functions

SYNOPSIS

```
#include <complex.h>

double complex catan(double complex z);
float complex catanf(float complex z);
long double complex catanl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex arc tangent of *z*, with branch cuts outside the interval [–*i*, +*i*] along the imaginary axis.

RETURN VALUE

These functions shall return the complex arc tangent value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [$-\pi/2$, $+\pi/2$] along the real axis.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ctan( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

SYNOPSIS

```
#include <complex.h>

double complex catanh(double complex z);
float complex catanhf(float complex z);
long double complex catanhl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic tangent of *z*, with branch cuts outside the interval [–1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic tangent value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the imaginary axis.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ctanh( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

5723 **NAME**

5724 catanl — complex arc tangent functions

5725 **SYNOPSIS**

5726
```
#include <complex.h>
```

5727
```
long double complex catanl(long double complex z);
```

5728 **DESCRIPTION**

5729 Refer to *catan*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

catclose — close a message catalog descriptor

SYNOPSIS

XSI
```
#include <nl_types.h>

int catclose(nl_catd catd);
```

DESCRIPTION

The *catclose*( ) function shall close the message catalog identified by *catd*. If a file descriptor is used to implement the type **nl_catd**, that file descriptor shall be closed.

RETURN VALUE

Upon successful completion, *catclose*( ) shall return 0; otherwise, –1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *catclose*( ) function may fail if:

[EBADF] The catalog descriptor is not valid.

[EINTR] The *catclose*( ) function was interrupted by a signal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catgets( ), *catopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<nl_types.h>**

CHANGE HISTORY

First released in Issue 2.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

catgets — read a program message

SYNOPSIS

XSI
```
#include <nl_types.h>

char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);
```

DESCRIPTION

The *catgets*( ) function shall attempt to read message *msg_id*, in set *set_id*, from the message catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an earlier call to *catopen*( ). The *s* argument points to a default message string which shall be returned by *catgets*( ) if it cannot retrieve the identified message.

The *catgets*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

If the identified message is retrieved successfully, *catgets*( ) shall return a pointer to an internal buffer area containing the null-terminated message string. If the call is unsuccessful for any reason, *s* shall be returned and *errno* may be set to indicate the error.

ERRORS

The *catgets*( ) function may fail if:

[EBADF] The *catd* argument is not a valid message catalog descriptor open for reading.

[EBADMSG] The message identified by *set_id* and *msg_id* in the specified message catalog did not satisfy implementation-defined security criteria.

[EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

[EINVAL] The message catalog identified by *catd* is corrupted.

[ENOMSG] The message identified by *set_id* and *msg_id* is not in the message catalog.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose( ), *catopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<nl_types.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 5

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 6

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

catopen — open a message catalog

SYNOPSIS

XSI

```
#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);
```

DESCRIPTION

The *catopen*( ) function shall open a message catalog and return a message catalog descriptor. The *name* argument specifies the name of the message catalog to be opened. If *name* contains a `'/'`, then *name* specifies a complete name for the message catalog. Otherwise, the environment variable *NLSPATH* is used with *name* substituted for the `%N` conversion specification (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If *NLSPATH* exists in the environment when the process starts, then if the process has appropriate privileges, the behavior of *catopen*( ) is undefined. If *NLSPATH* does not exist in the environment, or if a message catalog cannot be found in any of the components specified by *NLSPATH*, then an implementation-defined default path shall be used. This default may be affected by the setting of *LC_MESSAGES* if the value of *oflag* is NL_CAT_LOCALE, or the *LANG* environment variable if *oflag* is 0.

A message catalog descriptor shall remain valid in a process until that process closes it, or a successful call to one of the *exec* functions. A change in the setting of the *LC_MESSAGES* category may invalidate existing open catalogs.

If a file descriptor is used to implement message catalog descriptors, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the catalog without regard to the *LC_MESSAGES* category. If the *oflag* argument is NL_CAT_LOCALE, the *LC_MESSAGES* category is used to locate the message catalog (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables).

RETURN VALUE

Upon successful completion, *catopen*( ) shall return a message catalog descriptor for use on subsequent calls to *catgets*( ) and *catclose*( ). Otherwise, *catopen*( ) shall return (**nl_catd**) –1 and set *errno* to indicate the error.

ERRORS

The *catopen*( ) function may fail if:

[EACCES] Search permission is denied for the component of the path prefix of the message catalog or read permission is denied for the message catalog.

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENAMETOOLONG]
The length of a pathname of the message catalog exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

[ENOENT] The message catalog does not exist or the *name* argument points to an empty string.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[ENOMEM] Insufficient storage space is available.

[ENOTDIR] A component of the path prefix of the message catalog is not a directory.

EXAMPLES

None.

APPLICATION USAGE

Some implementations of *catopen*( ) use *malloc*( ) to allocate space for internal buffer areas. The *catopen*( ) function may fail if there is insufficient storage space available to accommodate these buffers.

Conforming applications must assume that message catalog descriptors are not valid after a call to one of the *exec* functions.

Application writers should be aware that guidelines for the location of message catalogs have not yet been developed. Therefore they should take care to avoid conflicting with catalogs used by other applications and the standard utilities.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose( ), *catgets*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<nl_types.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001

CHANGE HISTORY

First released in Issue 2.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

cbrt, cbrtf, cbrtl — cube root functions

SYNOPSIS

```
#include <math.h>

double cbrt(double x);
float cbrtf(float x);
long double cbrtl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the real cube root of their argument *x*.

RETURN VALUE

Upon successful completion, these functions shall return the cube root of *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

For some applications, a true cube root function, which returns negative results for negative arguments, is more appropriate than *pow*(*x*, 1.0/3.0), which returns a NaN for *x* less than 0.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *cbrt*( ) function is no longer marked as an extension.

The *cbrtf*( ) and *cbrtl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

ccos, ccosf, ccosl — complex cosine functions

SYNOPSIS

```
#include <complex.h>

double complex ccos(double complex z);
float complex ccosf(float complex z);
long double complex ccosl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex cosine of *z*.

RETURN VALUE

These functions shall return the complex cosine value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cacos( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

SYNOPSIS

```
#include <complex.h>

double complex ccosh(double complex z);
float complex ccoshf(float complex z);
long double complex ccoshl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex hyperbolic cosine of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic cosine value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cacosh( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5963 **NAME**

5964 ccosl — complex cosine functions

5965 **SYNOPSIS**

5966
```
#include <complex.h>
```

5967
```
long double complex ccosl(long double complex z);
```

5968 **DESCRIPTION**

5969 Refer to *ccos*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ceil, ceilf, ceill — ceiling value function

SYNOPSIS

```
#include <math.h>

double ceil(double x);
float ceilf(float x);
long double ceill(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the smallest integral value not less than *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, *ceil*( ), *ceilf*( ), and *ceill*( ) shall return the smallest integral value not less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

XSI If the correct value would cause overflow, a range error shall occur and *ceil*( ), *ceilf*( ), and *ceill*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

ERRORS

These functions shall fail if:

XSI

Range Error	The result overflows.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **int** or **long**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

The *ceil*( ) function can only overflow when the floating-point representation has DBL_MANT_DIG > DBL_MAX_EXP.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *floor*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *ceilf*( ) and *ceill*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

cexp, cexpf, cexpl — complex exponential functions

SYNOPSIS

```
#include <complex.h>

double complex cexp(double complex z);
float complex cexpf(float complex z);
long double complex cexpl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex exponent of *z*, defined as e^z.

RETURN VALUE

These functions shall return the complex exponential value of *z*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clog( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

cfgetispeed — get input baud rate

SYNOPSIS

```
#include <termios.h>

speed_t cfgetispeed(const struct termios *termios_p);
```

DESCRIPTION

The *cfgetispeed*( ) function shall extract the input baud rate from the **termios** structure to which the *termios_p* argument points.

This function shall return exactly the value in the **termios** data structure, without interpretation.

RETURN VALUE

Upon successful completion, *cfgetispeed*( ) shall return a value of type **speed_t** representing the input baud rate.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The term ‘‘baud’’ is used historically here, but is not technically correct. This is properly ‘‘bits per second’’, which may not be the same as baud. However, the term is used because of the historical usage and understanding.

The *cfgetospeed*( ), *cfgetispeed*( ), *cfsetospeed*( ), and *cfsetispeed*( ) functions do not take arguments as numbers, but rather as symbolic names. There are two reasons for this:

1. Historically, numbers were not used because of the way the rate was stored in the data structure. This is retained even though a function is now used.
2. More importantly, only a limited set of possible rates is at all portable, and this constrains the application to that set.

There is nothing to prevent an implementation accepting as an extension a number (such as 126), and since the encoding of the Bxxx symbols is not specified, this can be done to avoid introducing ambiguity.

Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications in this volume of IEEE Std 1003.1-2001 have made it possible to determine whether split rates are supported and to support them without having to treat zero as a special case. Since this functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is the literal constant 0, not the symbolic constant B0. This volume of IEEE Std 1003.1-2001 does not preclude B0 from being defined as the value 0; in fact, implementations would likely benefit from the two being equivalent. This volume of IEEE Std 1003.1-2001 does not fully specify whether the previous *cfsetispeed*( ) value is retained after a *tcgetattr*( ) as the actual value or as zero. Therefore, conforming applications should always set both the input speed and output speed when setting either.

In historical implementations, the baud rate information is traditionally kept in **c_cflag**. Applications should be written to presume that this might be the case (and thus not blindly copy **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag** field absolutely after setting a baud rate is a non-portable action because of this. In general, the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

unused parts of the flag fields might be used by the implementation and should not be blindly copied from the descriptions of one terminal device to another.

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetospeed( ), *cfsetispeed*( ), *cfsetospeed*( ), *tcgetattr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

cfgetospeed — get output baud rate

SYNOPSIS

```
#include <termios.h>

speed_t cfgetospeed(const struct termios *termios_p);
```

DESCRIPTION

The *cfgetospeed*( ) function shall extract the output baud rate from the **termios** structure to which the *termios_p* argument points.

This function shall return exactly the value in the **termios** data structure, without interpretation.

RETURN VALUE

Upon successful completion, *cfgetospeed*( ) shall return a value of type **speed_t** representing the output baud rate.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfsetispeed*( ), *cfsetospeed*( ), *tcgetattr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

cfsetispeed — set input baud rate

SYNOPSIS

```
#include <termios.h>

int cfsetispeed(struct termios *termios_p, speed_t speed);
```

DESCRIPTION

The *cfsetispeed*( ) function shall set the input baud rate stored in the structure pointed to by *termios_p* to *speed*.

There shall be no effect on the baud rates set in the hardware until a subsequent successful call to *tcsetattr*( ) with the same **termios** structure. Similarly, errors resulting from attempts to set baud rates not supported by the terminal device need not be detected until the *tcsetattr*( ) function is called.

RETURN VALUE

Upon successful completion, *cfsetispeed*( ) shall return 0; otherwise, −1 shall be returned, and *errno* may be set to indicate the error.

ERRORS

The *cfsetispeed*( ) function may fail if:

[EINVAL] The *speed* value is not a valid baud rate.

[EINVAL] The value of *speed* is outside the range of possible speed values as specified in **<termios.h>**.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfgetospeed*( ), *cfsetospeed*( ), *tcsetattr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* and the [EINVAL] error conditions are added.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

cfsetospeed — set output baud rate

SYNOPSIS

```
#include <termios.h>

int cfsetospeed(struct termios *termios_p, speed_t speed);
```

DESCRIPTION

The *cfsetospeed*( ) function shall set the output baud rate stored in the structure pointed to by *termios_p* to *speed*.

There shall be no effect on the baud rates set in the hardware until a subsequent successful call to *tcsetattr*( ) with the same **termios** structure. Similarly, errors resulting from attempts to set baud rates not supported by the terminal device need not be detected until the *tcsetattr*( ) function is called.

RETURN VALUE

Upon successful completion, *cfsetospeed*( ) shall return 0; otherwise, it shall return –1 and *errno* may be set to indicate the error.

ERRORS

The *cfsetospeed*( ) function may fail if:

[EINVAL] The *speed* value is not a valid baud rate.

[EINVAL] The value of *speed* is outside the range of possible speed values as specified in **<termios.h>**.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfgetospeed*( ), *cfsetispeed*( ), *tcsetattr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* and the [EINVAL] error conditions are added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

chdir — change working directory

SYNOPSIS

```
#include <unistd.h>

int chdir(const char *path);
```

DESCRIPTION

The *chdir*( ) function shall cause the directory named by the pathname pointed to by the *path* argument to become the current working directory; that is, the starting point for path searches for pathnames not beginning with '/'.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned, the current working directory shall remain unchanged, and *errno* shall be set to indicate the error.

ERRORS

The *chdir*( ) function shall fail if:

[EACCES] Search permission is denied for any component of the pathname.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing directory or *path* is an empty string.

[ENOTDIR] A component of the pathname is not a directory.

The *chdir*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

EXAMPLES

Changing the Current Working Directory

The following example makes the value pointed to by **directory**, **/tmp**, the current working directory.

```
#include <unistd.h>
...
char *directory = "/tmp";
int ret;

ret = chdir (directory);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

APPLICATION USAGE

None.

RATIONALE

The *chdir*( ) function only affects the working directory of the current process.

FUTURE DIRECTIONS

None.

SEE ALSO

getcwd( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The APPLICATION USAGE section is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

chmod — change mode of a file

SYNOPSIS

```
#include <sys/stat.h>

int chmod(const char *path, mode_t mode);
```

DESCRIPTION

XSI The *chmod*( ) function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of the file named by the pathname pointed to by the *path* argument to the corresponding bits in the *mode* argument. The application shall ensure that the effective user ID of the process matches the owner of the file or the process has appropriate privileges in order to do this.

XSI S_ISUID, S_ISGID, S_ISVTX, and the file permission bits are described in **<sys/stat.h>**.

If the calling process does not have appropriate privileges, and if the group ID of the file does not match the effective group ID or one of the supplementary group IDs and if the file is a regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon successful return from *chmod*( ).

Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in *mode* to be ignored.

The effect on file descriptors for files open at the time of a call to *chmod*( ) is implementation-defined.

Upon successful completion, *chmod*( ) shall mark for update the *st_ctime* field of the file.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error. If –1 is returned, no change to the file mode occurs.

ERRORS

The *chmod*( ) function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOTDIR] A component of the path prefix is not a directory.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[EPERM] The effective user ID does not match the owner of the file and the process does not have appropriate privileges.

[EROFS] The named file resides on a read-only file system.

The *chmod*( ) function may fail if:

[EINTR] A signal was caught during execution of the function.

[EINVAL] The value of the *mode* argument is invalid.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname strings exceeded {PATH_MAX}.

EXAMPLES

Setting Read Permissions for User, Group, and Others

The following example sets read permissions for the owner, group, and others.

```
#include <sys/stat.h>

const char *path;
...
chmod(path, S_IRUSR|S_IRGRP|S_IROTH);
```

Setting Read, Write, and Execute Permissions for the Owner Only

The following example sets read, write, and execute permissions for the owner, and no permissions for group and others.

```
#include <sys/stat.h>

const char *path;
...
chmod(path, S_IRWXU);
```

Setting Different Permissions for Owner, Group, and Other

The following example sets owner permissions for CHANGEFILE to read, write, and execute, group permissions to read and execute, and other permissions to read.

```
#include <sys/stat.h>

#define CHANGEFILE "/etc/myfile"
...
chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

Setting and Checking File Permissions

The following example sets the file permission bits for a file named **/home/cnd/mod1**, then calls the *stat*( ) function to verify the permissions.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
struct stat buffer
...
chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
status = stat("home/cnd/mod1", &buffer;);
```

APPLICATION USAGE

In order to ensure that the S_ISUID and S_ISGID bits are set, an application requiring this should use *stat*( ) after a successful *chmod*( ) to verify this.

Any file descriptors currently open by any process on the file could possibly become invalid if the mode of the file is changed to a value which would deny access to that process. One

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

situation where this could occur is on a stateless file system. This behavior will not occur in a conforming environment.

RATIONALE

This volume of IEEE Std 1003.1-2001 specifies that the S_ISGID bit is cleared by *chmod*( ) on a regular file under certain conditions. This is specified on the assumption that regular files may be executed, and the system should prevent users from making executable *setgid*( ) files perform with privileges that the caller does not have. On implementations that support execution of other file types, the S_ISGID bit should be cleared for those file types under the same circumstances.

Implementations that use the S_ISUID bit to indicate some other function (for example, mandatory record locking) on non-executable files need not clear this bit on writing. They should clear the bit for executable files and any other cases where the bit grants special powers to processes that change the file contents. Similar comments apply to the S_ISGID bit.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( ), *mkdir*( ), *mkfifo*( ), *open*( ), *stat*( ), *statvfs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EINVAL] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6378 **NAME**

6379 chown — change owner and group of a file

6380 **SYNOPSIS**

6381 `#include <unistd.h>`

6382 `int chown(const char *path, uid_t owner, gid_t group);`

6383 **DESCRIPTION**

6384 The *chown*( ) function shall change the user and group ownership of a file.

6385 The *path* argument points to a pathname naming a file. The user ID and group ID of the named
6386 file shall be set to the numeric values contained in *owner* and *group*, respectively.

6387 Only processes with an effective user ID equal to the user ID of the file or with appropriate
6388 privileges may change the ownership of a file. If _POSIX_CHOWN_RESTRICTED is in effect for
6389 *path*:

6390 • Changing the user ID is restricted to processes with appropriate privileges.

6391 • Changing the group ID is permitted to a process with an effective user ID equal to the user
6392 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's user
6393 ID or (**uid_t**)–1 and *group* is equal either to the calling process' effective group ID or to one of
6394 its supplementary group IDs.

6395 If the specified file is a regular file, one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of
6396 the file mode are set, and the process does not have appropriate privileges, the set-user-ID
6397 (S_ISUID) and set-group-ID (S_ISGID) bits of the file mode shall be cleared upon successful
6398 return from *chown*( ). If the specified file is a regular file, one or more of the S_IXUSR, S_IXGRP,
6399 or S_IXOTH bits of the file mode are set, and the process has appropriate privileges, it is
6400 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown*( )
6401 function is successfully invoked on a file that is not a regular file and one or more of the
6402 S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set, the set-user-ID and set-group-ID
6403 bits may be cleared.

6404 If *owner* or *group* is specified as (**uid_t**)–1 or (**gid_t**)–1, respectively, the corresponding ID of the
6405 file shall not be changed. If both owner and group are –1, the times need not be updated.

6406 Upon successful completion, *chown*( ) shall mark for update the *st_ctime* field of the file.

6407 **RETURN VALUE**

6408 Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to
6409 indicate the error. If –1 is returned, no changes are made in the user ID and group ID of the file.

6410 **ERRORS**

6411 The *chown*( ) function shall fail if:

6412 [EACCES] Search permission is denied on a component of the path prefix.

6413 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
6414 argument.

6415 [ENAMETOOLONG]
6416 The length of the *path* argument exceeds {PATH_MAX} or a pathname
6417 component is longer than {NAME_MAX}.

6418 [ENOTDIR] A component of the path prefix is not a directory.

6419 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EPERM] The effective user ID does not match the owner of the file, or the calling process does not have appropriate privileges and _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

[EROFS] The named file resides on a read-only file system.

The *chown*( ) function may fail if:

[EIO] An I/O error occurred while reading or writing to the file system.

[EINTR] The *chown*( ) function was interrupted by a signal which was caught.

[EINVAL] The owner or group ID supplied is not a value supported by the implementation.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

EXAMPLES

None.

APPLICATION USAGE

Although *chown*( ) can be used on some implementations by the file owner to change the owner and group to any desired values, the only portable use of this function is to change the group of a file to the effective GID of the calling process or to a member of its group set.

RATIONALE

System III and System V allow a user to give away files; that is, the owner of a file may change its user ID to anything. This is a serious problem for implementations that are intended to meet government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the user ID of a file. Some government agencies (usually not ones concerned directly with security) find this limitation too confining. This volume of IEEE Std 1003.1-2001 uses *may* to permit secure implementations while not disallowing System V.

System III and System V allow the owner of a file to change the group ID to anything. Version 7 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to change the group ID of a file to its effective group ID or to any of the groups in the list of supplementary group IDs, but to no others.

The POSIX.1-1990 standard requires that the *chown*( ) function invoked by a non-appropriate privileged process clear the S_ISGID and the S_ISUID bits for regular files, and permits them to be cleared for other types of files. This is so that changes in accessibility do not accidentally cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-executable data files also clears the mandatory file locking bit (shared with S_ISGID), which is an extension on many implementations (it first appeared in System V). These bits should only be required to be cleared on regular files that have one or more of their execute bits set.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *pathconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The wording describing the optional dependency on _POSIX_CHOWN_RESTRICTED is restored.
- The [EPERM] error is restored as an error dependent on _POSIX_CHOWN_RESTRICTED. This is since its operand is a pathname and applications should be aware that the error may not occur for that pathname if the file system does not support _POSIX_CHOWN_RESTRICTED.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The value for *owner* of (**uid_t**)–1 allows the use of –1 by the owner of a file to change the group ID only. A corresponding change is made for group.
- The [ELOOP] mandatory error condition is added.
- The [EIO] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that the S_ISUID and S_ISGID bits do not need to be cleared when the process has appropriate privileges.
- The [ELOOP] optional error condition is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

cimag, cimagf, cimagl — complex imaginary functions

SYNOPSIS

```
#include <complex.h>

double cimag(double complex z);
float cimagf(float complex z);
long double cimagl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the imaginary part of *z*.

RETURN VALUE

These functions shall return the imaginary part value (as a real).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

For a variable *z* of complex type:

```
z == creal(z) + cimag(z)*I
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *conj*( ), *cproj*( ), *creal*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

clearerr — clear indicators on a stream

SYNOPSIS

```
#include <stdio.h>

void clearerr(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *clearerr*( ) function shall clear the end-of-file and error indicators for the stream to which *stream* points.

RETURN VALUE

The *clearerr*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

clock — report CPU time used

SYNOPSIS

```
#include <time.h>

clock_t clock(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *clock*( ) function shall return the implementation's best approximation to the processor time used by the process since the beginning of an implementation-defined era related only to the process invocation.

RETURN VALUE

To determine the time in seconds, the value returned by *clock*( ) should be divided by the value XSI of the macro CLOCKS_PER_SEC. CLOCKS_PER_SEC is defined to be one million in **<time.h>**. If the processor time used is not available or its value cannot be represented, the function shall return the value (**clock_t**)–1.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

In order to measure the time spent in a program, *clock*( ) should be called at the start of the program and its return value subtracted from the value returned by subsequent calls. The value returned by *clock*( ) is defined for compatibility across systems that have clocks with different resolutions. The resolution on any particular system need not be to microsecond accuracy.

The value returned by *clock*( ) may wrap around on some implementations. For example, on a machine with 32-bit values for **clock_t**, it wraps after 2 147 seconds or 36 minutes.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *ctime*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)

SYNOPSIS

CPT
```
#include <time.h>

int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

DESCRIPTION

The *clock_getcpuclockid*( ) function shall return the clock ID of the CPU-time clock of the process specified by *pid*. If the process described by *pid* exists and the calling process has permission, the clock ID of this clock shall be returned in *clock_id*.

If *pid* is zero, the *clock_getcpuclockid*( ) function shall return the clock ID of the CPU-time clock of the process making the call, in *clock_id*.

The conditions under which one process has permission to obtain the CPU-time clock ID of other processes are implementation-defined.

RETURN VALUE

Upon successful completion, *clock_getcpuclockid*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *clock_getcpuclockid*( ) function shall fail if:

[EPERM] The requesting process does not have permission to access the CPU-time clock for the process.

The *clock_getcpuclockid*( ) function may fail if:

[ESRCH] No process can be found corresponding to the process specified by *pid*.

EXAMPLES

None.

APPLICATION USAGE

The *clock_getcpuclockid*( ) function is part of the Process CPU-Time Clocks option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *timer_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

clock_getres, clock_gettime, clock_settime — clock and timer functions (**REALTIME**)

SYNOPSIS

TMR
```
#include <time.h>

int clock_getres(clockid_t clock_id, struct timespec *res);
int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

DESCRIPTION

The *clock_getres*( ) function shall return the resolution of any clock. Clock resolutions are implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL, the clock resolution is not returned. If the *time* argument of *clock_settime*( ) is not a multiple of *res*, then the value is truncated to a multiple of *res*.

The *clock_gettime*( ) function shall return the current value *tp* for the specified clock, *clock_id*.

The *clock_settime*( ) function shall set the specified clock, *clock_id*, to the value specified by *tp*. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock shall be truncated down to the smaller multiple of the resolution.

A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). All implementations shall support a *clock_id* of CLOCK_REALTIME as defined in **<time.h>**. This clock represents the realtime clock for the system. For this clock, the values returned by *clock_gettime*( ) and specified by *clock_settime*( ) represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation may also support additional clocks. The interpretation of time values for these clocks is unspecified.

If the value of the CLOCK_REALTIME clock is set via *clock_settime*( ), the new value of the clock shall be used to determine the time of expiration for absolute time services based upon the CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the absolute time requested at the invocation of such a time service is before the new value of the clock, the time service shall expire immediately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via *clock_settime*( ) shall have no effect on threads that are blocked waiting for a relative time service based upon this clock, including the *nanosleep*( ) function; nor on the expiration of relative timers based upon this clock. Consequently, these time services shall expire when the requested relative interval elapses, independently of the new or old value of the clock.

MON If the Monotonic Clock option is supported, all implementations shall support a *clock_id* of CLOCK_MONOTONIC defined in **<time.h>**. This clock represents the monotonic clock for the system. For this clock, the value returned by *clock_gettime*( ) represents the amount of time (in seconds and nanoseconds) since an unspecified point in the past (for example, system start-up time, or the Epoch). This point does not change after system start-up time. The value of the CLOCK_MONOTONIC clock cannot be set via *clock_settime*( ). This function shall fail if it is invoked with a *clock_id* argument of CLOCK_MONOTONIC.

The effect of setting a clock via *clock_settime*( ) on armed per-process timers associated with a clock other than CLOCK_REALTIME is implementation-defined.

CS If the value of the CLOCK_REALTIME clock is set via *clock_settime*( ), the new value of the clock shall be used to determine the time at which the system shall awaken a thread blocked on an

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

absolute *clock_nanosleep*( ) call based upon the CLOCK_REALTIME clock. If the absolute time requested at the invocation of such a time service is before the new value of the clock, the call shall return immediately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via *clock_settime*( ) shall have no effect on any thread that is blocked on a relative *clock_nanosleep*( ) call. Consequently, the call shall return when the requested relative interval elapses, independently of the new or old value of the clock.

The appropriate privilege to set a particular clock is implementation-defined.

CPT If _POSIX_CPUTIME is defined, implementations shall support clock ID values obtained by invoking *clock_getcpuclockid*( ), which represent the CPU-time clock of a given process. Implementations shall also support the special **clockid_t** value CLOCK_PROCESS_CPUTIME_ID, which represents the CPU-time clock of the calling process when invoking one of the *clock_**( ) or *timer_**( ) functions. For these clock IDs, the values returned by *clock_gettime*( ) and specified by *clock_settime*( ) represent the amount of execution time of the process associated with the clock. Changing the value of a CPU-time clock via *clock_settime*( ) shall have no effect on the behavior of the sporadic server scheduling policy (see **Scheduling Policies** (on page 44)).

TCT If _POSIX_THREAD_CPUTIME is defined, implementations shall support clock ID values obtained by invoking *pthread_getcpuclockid*( ), which represent the CPU-time clock of a given thread. Implementations shall also support the special **clockid_t** value CLOCK_THREAD_CPUTIME_ID, which represents the CPU-time clock of the calling thread when invoking one of the *clock_**( ) or *timer_**( ) functions. For these clock IDs, the values returned by *clock_gettime*( ) and specified by *clock_settime*( ) shall represent the amount of execution time of the thread associated with the clock. Changing the value of a CPU-time clock via *clock_settime*( ) shall have no effect on the behavior of the sporadic server scheduling policy (see **Scheduling Policies** (on page 44)).

RETURN VALUE

A return value of 0 shall indicate that the call succeeded. A return value of –1 shall indicate that an error occurred, and *errno* shall be set to indicate the error.

ERRORS

The *clock_getres*( ), *clock_gettime*( ), and *clock_settime*( ) functions shall fail if:

[EINVAL] The *clock_id* argument does not specify a known clock.

The *clock_settime*( ) function shall fail if:

[EINVAL] The *tp* argument to *clock_settime*( ) is outside the range for the given clock ID.

[EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

MON [EINVAL] The value of the *clock_id* argument is CLOCK_MONOTONIC.

The *clock_settime*( ) function may fail if:

[EPERM] The requesting process does not have the appropriate privilege to set the specified clock.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Timers option and need not be available on all implementations.

Note that the absolute value of the monotonic clock is meaningless (because its origin is arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the fact that the value of this clock is never set and, therefore, that time intervals measured with this clock will not be affected by calls to *clock_settime*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getcpuclockid( ), *clock_nanosleep*( ), *ctime*( ), *mq_timedreceive*( ), *mq_timedsend*( ), *nanosleep*( ), *pthread_mutex_timedlock*( ), *sem_timedwait*( ), *time*( ), *timer_create*( ), *timer_getoverrun*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

The APPLICATION USAGE section is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added of the effect of resetting the clock resolution.

CPU-time clocks and the *clock_getcpuclockid*( ) function are added for alignment with IEEE Std 1003.1d-1999.

The following changes are added for alignment with IEEE Std 1003.1j-2000:

- The DESCRIPTION is updated as follows:
  - The value returned by *clock_gettime*( ) for CLOCK_MONOTONIC is specified.
  - The *clock_settime*( ) function failing for CLOCK_MONOTONIC is specified.
  - The effects of *clock_settime*( ) on the *clock_nanosleep*( ) function with respect to CLOCK_REALTIME are specified.
- An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime*( ) fails for CLOCK_MONOTONIC.
- The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not and shall not be set by *clock_settime*( ) since the absolute value of the CLOCK_MONOTONIC clock is meaningless.
- The *clock_nanosleep*( ), *mq_timedreceive*( ), *mq_timedsend*( ), *pthread_mutex_timedlock*( ), *sem_timedwait*( ), *timer_create*( ), and *timer_settime*( ) functions are added to the SEE ALSO section.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

clock_nanosleep — high resolution sleep with specifiable clock (**ADVANCED REALTIME**)

SYNOPSIS

CS

```
#include <time.h>

int clock_nanosleep(clockid_t clock_id, int flags,
    const struct timespec *rqtp, struct timespec *rmtp);
```

DESCRIPTION

If the flag TIMER_ABSTIME is not set in the *flags* argument, the *clock_nanosleep*( ) function shall cause the current thread to be suspended from execution until either the time interval specified by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to invoke a signal-catching function, or the process is terminated. The clock used to measure the time shall be the clock specified by *clock_id*.

If the flag TIMER_ABSTIME is set in the *flags* argument, the *clock_nanosleep*( ) function shall cause the current thread to be suspended from execution until either the time value of the clock specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is delivered to the calling thread and its action is to invoke a signal-catching function, or the process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or equal to the time value of the specified clock, then *clock_nanosleep*( ) shall return immediately and the calling process shall not be suspended.

The suspension time caused by this function may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution, or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time for the relative *clock_nanosleep*( ) function (that is, with the TIMER_ABSTIME flag not set) shall not be less than the time interval specified by *rqtp*, as measured by the corresponding clock. The suspension for the absolute *clock_nanosleep*( ) function (that is, with the TIMER_ABSTIME flag set) shall be in effect at least until the value of the corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being interrupted by a signal.

The use of the *clock_nanosleep*( ) function shall have no effect on the action or blockage of any signal.

The *clock_nanosleep*( ) function shall fail if the *clock_id* argument refers to the CPU-time clock of the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are allowed.

RETURN VALUE

If the *clock_nanosleep*( ) function returns because the requested time has elapsed, its return value shall be zero.

If the *clock_nanosleep*( ) function returns because it has been interrupted by a signal, it shall return the corresponding error value. For the relative *clock_nanosleep*( ) function, if the *rmtp* argument is non-NULL, the **timespec** structure referenced by it shall be updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). If the *rmtp* argument is NULL, the remaining time is not returned. The absolute *clock_nanosleep*( ) function has no effect on the structure referenced by *rmtp*.

If *clock_nanosleep*( ) fails, it shall return the corresponding error value.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ERRORS

The *clock_nanosleep*( ) function shall fail if:

[EINTR] The *clock_nanosleep*( ) function was interrupted by a signal.

[EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million; or the TIMER_ABSTIME flag was specified in flags and the *rqtp* argument is outside the range for the clock specified by *clock_id*; or the *clock_id* argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.

[ENOTSUP] The *clock_id* argument specifies a clock for which *clock_nanosleep*( ) is not supported, such as a CPU-time clock.

EXAMPLES

None.

APPLICATION USAGE

Calling *clock_nanosleep*( ) with the value TIMER_ABSTIME not set in the *flags* argument and with a *clock_id* of CLOCK_REALTIME is equivalent to calling *nanosleep*( ) with the same *rqtp* and *rmtp* arguments.

RATIONALE

The *nanosleep*( ) function specifies that the system-wide clock CLOCK_REALTIME is used to measure the elapsed time for this time service. However, with the introduction of the monotonic clock CLOCK_MONOTONIC a new relative sleep function is needed to allow an application to take advantage of the special characteristics of this clock.

There are many applications in which a process needs to be suspended and then activated multiple times in a periodic way; for example, to poll the status of a non-interrupting device or to refresh a display device. For these cases, it is known that precise periodic activation cannot be achieved with a relative *sleep*( ) or *nanosleep*( ) function call. Suppose, for example, a periodic process that is activated at time $T0$, executes for a while, and then wants to suspend itself until time $T0+T$, the period being T. If this process wants to use the *nanosleep*( ) function, it must first call *clock_gettime*( ) to get the current time, then calculate the difference between the current time and $T0+T$ and, finally, call *nanosleep*( ) using the computed interval. However, the process could be preempted by a different process between the two function calls, and in this case the interval computed would be wrong; the process would wake up later than desired. This problem would not occur with the absolute *clock_nanosleep*( ) function, since only one function call would be necessary to suspend the process until the desired time. In other cases, however, a relative sleep is needed, and that is why both functionalities are required.

Although it is possible to implement periodic processes using the timers interface, this implementation would require the use of signals, and the reservation of some signal numbers. In this regard, the reasons for including an absolute version of the *clock_nanosleep*( ) function in IEEE Std 1003.1-2001 are the same as for the inclusion of the relative *nanosleep*( ).

It is also possible to implement precise periodic processes using *pthread_cond_timedwait*( ), in which an absolute timeout is specified that takes effect if the condition variable involved is never signaled. However, the use of this interface is unnatural, and involves performing other operations on mutexes and condition variables that imply an unnecessary overhead. Furthermore, *pthread_cond_timedwait*( ) is not available in implementations that do not support threads.

Although the interface of the relative and absolute versions of the new high resolution sleep service is the same *clock_nanosleep*( ) function, the *rmtp* argument is only used in the relative sleep. This argument is needed in the relative *clock_nanosleep*( ) function to reissue the function

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep*( ) function call; if the call is interrupted by a signal, the absolute *clock_nanosleep*( ) function can be invoked again with the same *rqtp* argument used in the interrupted call.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *nanosleep*( ), *pthread_cond_timedwait*( ), *sleep*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

clock_settime — clock and timer functions (**REALTIME**)

SYNOPSIS

TMR

```
#include <time.h>

int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

DESCRIPTION

Refer to *clock_getres*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

clog, clogf, clogl — complex natural logarithm functions

SYNOPSIS

```
#include <complex.h>

double complex clog(double complex z);
float complex clogf(float complex z);
long double complex clogl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex natural (base *e*) logarithm of *z*, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the complex natural logarithm value, in the range of a strip mathematically unbounded along the real axis and in the interval [$-i\pi$, $+i\pi$] along the imaginary axis.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cexp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

close — close a file descriptor

SYNOPSIS

```
#include <unistd.h>

int close(int fildes);
```

DESCRIPTION

The *close*( ) function shall deallocate the file descriptor indicated by *fildes*. To deallocate means to make the file descriptor available for return by subsequent calls to *open*( ) or other functions that allocate file descriptors. All outstanding record locks owned by the process on the file associated with the file descriptor shall be removed (that is, unlocked).

If *close*( ) is interrupted by a signal that is to be caught, it shall return –1 with *errno* set to [EINTR] and the state of *fildes* is unspecified. If an I/O error occurred while reading from or writing to the file system during *close*( ), it may return –1 with *errno* set to [EIO]; if this error is returned, the state of *fildes* is unspecified.

When all file descriptors associated with a pipe or FIFO special file are closed, any data remaining in the pipe or FIFO shall be discarded.

When all file descriptors associated with an open file description have been closed, the open file description shall be freed.

If the link count of the file is 0, when all file descriptors associated with the file are closed, the space occupied by the file shall be freed and the file shall no longer be accessible.

XSR If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive a SIGPOLL signal for events associated with that STREAM, the calling process shall be unregistered for events associated with the STREAM. The last *close*( ) for a STREAM shall cause the STREAM associated with *fildes* to be dismantled. If O_NONBLOCK is not set and there have been no signals posted for the STREAM, and if there is data on the module's write queue, *close*( ) shall wait for an unspecified time (for each module and driver) for any output to drain before dismantling the STREAM. The time delay can be changed via an I_SETCLTIME *ioctl*( ) request. If the O_NONBLOCK flag is set, or if there are any pending signals, *close*( ) shall not wait for output to drain, and shall dismantle the STREAM immediately.

If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a pipe, the last *close*( ) shall cause a hangup to occur on the other end of the pipe. In addition, if the other end of the pipe has been named by *fattach*( ), then the last *close*( ) shall force the named end to be detached by *fdetach*( ). If the named end has no open file descriptors associated with it and gets detached, the STREAM associated with that end shall also be dismantled.

XSI If *fildes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal flushes all queued input and output.

XSR If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message may be sent to the master.

AIO When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close*( ) is called, that I/O operation may be canceled. An I/O operation that is not canceled completes as if the *close*( ) operation had not yet occurred. All operations that are not canceled shall complete as if the *close*( ) blocked until the operations completed. The *close*( ) operation itself need not block awaiting such I/O completion. Whether any I/O operation is canceled, and which I/O operation may be canceled upon *close*( ), is implementation-defined.

-`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

MF|SHM If a shared memory object or a memory mapped file remains referenced at the last close (that is, a process has it mapped), then the entire contents of the memory object shall persist until the memory object becomes unreferenced. If this is the last close of a shared memory object or a memory mapped file and the close results in the memory object becoming unreferenced, and the memory object has been unlinked, then the memory object shall be removed.

If *fildes* refers to a socket, *close*( ) shall cause the socket to be destroyed. If the socket is in connection-mode, and the SO_LINGER option is set for the socket with non-zero linger time, and the socket has untransmitted data, then *close*( ) shall block for up to the current linger interval until all data is transmitted.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *close*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINTR] The *close*( ) function was interrupted by a signal.

The *close*( ) function may fail if:

[EIO] An I/O error occurred while reading from or writing to the file system.

EXAMPLES

Reassigning a File Descriptor

The following example closes the file descriptor associated with standard output for the current process, re-assigns standard output to a new file descriptor, and closes the original file descriptor to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard input) is not closed.

```
#include <unistd.h>
...
int pfd;
...
close(1);
dup(pfd);
close(pfd);
...
```

Incidentally, this is exactly what could be achieved using:

```
dup2(pfd, 1);
close(pfd);
```

Closing a File Descriptor

In the following example, *close*( ) is used to close a file descriptor after an unsuccessful attempt is made to associate that file descriptor with a stream.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
#define LOCKFILE "/etc/ptmp"
...
int pfd;
FILE *fpfd;
...
if ((fpfd = fdopen (pfd, "w")) == NULL) {
    close(pfd);
    unlink(LOCKFILE);
    exit(1);
}
...
```

APPLICATION USAGE

An application that had used the *stdio* routine *fopen*( ) to open a file should use the corresponding *fclose*( ) routine rather than *close*( ). Once a file is closed, the file descriptor no longer exists, since the integer corresponding to it no longer refers to a file.

RATIONALE

The use of interruptible device close routines should be discouraged to avoid problems with the implicit closes of file descriptors by *exec* and *exit*( ). This volume of IEEE Std 1003.1-2001 only intends to permit such behavior by specifying the [EINTR] error condition.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 38), *fattach*( ), *fclose*( ), *fdetach*( ), *fopen*( ), *ioctl*( ), *open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Issue 6

The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error condition is added as an optional error.
- The DESCRIPTION is updated to describe the state of the *fildes* file descriptor as unspecified if an I/O error occurs and an [EIO] error condition is returned.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that shared memory objects and memory mapped files (and not typed memory objects) are the types of memory objects to which the paragraph on last closes applies.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded text relating to the master side of a pseudo-terminal. The reason for the change is that the behavior of pseudo-terminals and regular terminals should be as much alike as possible in this case; the change achieves that and matches historical behavior.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

closedir — close a directory stream

SYNOPSIS

```
#include <dirent.h>

int closedir(DIR *dirp);
```

DESCRIPTION

The *closedir*( ) function shall close the directory stream referred to by the argument *dirp*. Upon return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file descriptor is used to implement type **DIR**, that file descriptor shall be closed.

RETURN VALUE

Upon successful completion, *closedir*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *closedir*( ) function may fail if:

[EBADF] The *dirp* argument does not refer to an open directory stream.

[EINTR] The *closedir*( ) function was interrupted by a signal.

EXAMPLES

Closing a Directory Stream

The following program fragment demonstrates how the *closedir*( ) function is used.

```
...
    DIR *dir;
    struct dirent *dp;
...
    if ((dir = opendir (".")) == NULL) {
...
    }
    while ((dp = readdir (dir)) != NULL) {
...
    }
    closedir(dir);
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

opendir( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

7053 **CHANGE HISTORY**

7054 First released in Issue 2.

7055 **Issue 6**

7056 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

7057 The following new requirements on POSIX implementations derive from alignment with the
7058 Single UNIX Specification:

7059 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
7060 required for conforming implementations of previous POSIX specifications, it was not
7061 required for UNIX applications.

7062 • The [EINTR] error condition is added as an optional error condition.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7063 **NAME**

7064 closelog, openlog, setlogmask, syslog — control system log

7065 **SYNOPSIS**

7066 XSI
```
#include <syslog.h>

void closelog(void);
void openlog(const char *ident, int logopt, int facility);
int setlogmask(int maskpri);
void syslog(int priority, const char *message, ... /* arguments */);
```
7071

7072 **DESCRIPTION**

7073 The *syslog*( ) function shall send a message to an implementation-defined logging facility, which
7074 may log it in an implementation-defined system log, write it to the system console, forward it to
7075 a list of users, or forward it to the logging facility on another host over the network. The logged
7076 message shall include a message header and a message body. The message header contains at
7077 least a timestamp and a tag string.

7078 The message body is generated from the *message* and following arguments in the same manner
7079 as if these were arguments to *printf*( ), except that the additional conversion specification `%m`
7080 shall be recognized; it shall convert no arguments, shall cause the output of the error message
7081 string associated with the value of *errno* on entry to *syslog*( ), and may be mixed with argument
7082 specifications of the `"%n$"` form. If a complete conversion specification with the `m` conversion
7083 specifier character is not just `%m`, the behavior is undefined. A trailing <newline> may be added
7084 if needed.

7085 Values of the *priority* argument are formed by OR'ing together a severity-level value and an
7086 optional facility value. If no facility value is specified, the current default facility value is used.

7087 Possible values of severity level include:

7088 LOG_EMERG A panic condition.

7089 LOG_ALERT A condition that should be corrected immediately, such as a corrupted system
7090 database.

7091 LOG_CRIT Critical conditions, such as hard device errors.

7092 LOG_ERR Errors.

7093 LOG_WARNING
7094 Warning messages.

7095 LOG_NOTICE Conditions that are not error conditions, but that may require special
7096 handling.

7097 LOG_INFO Informational messages.

7098 LOG_DEBUG Messages that contain information normally of use only when debugging a
7099 program.

7100 The facility indicates the application or system component generating the message. Possible
7101 facility values include:

7102 LOG_USER Messages generated by arbitrary processes. This is the default facility
7103 identifier if none is specified.

7104 LOG_LOCAL0 Reserved for local use.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

LOG_LOCAL1 Reserved for local use.

LOG_LOCAL2 Reserved for local use.

LOG_LOCAL3 Reserved for local use.

LOG_LOCAL4 Reserved for local use.

LOG_LOCAL5 Reserved for local use.

LOG_LOCAL6 Reserved for local use.

LOG_LOCAL7 Reserved for local use.

The *openlog*( ) function shall set process attributes that affect subsequent calls to *syslog*( ). The *ident* argument is a string that is prepended to every message. The *logopt* argument indicates logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of the following:

LOG_PID Log the process ID with each message. This is useful for identifying specific processes.

LOG_CONS Write messages to the system console if they cannot be sent to the logging facility. The *syslog*( ) function ensures that the process does not acquire the console as a controlling terminal in the process of writing the message.

LOG_NDELAY Open the connection to the logging facility immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.

LOG_ODELAY Delay open until *syslog*( ) is called.

LOG_NOWAIT Do not wait for child processes that may have been created during the course of logging the message. This option should be used by processes that enable notification of child termination using SIGCHLD, since *syslog*( ) may otherwise block waiting for a child whose exit status has already been collected.

The *facility* argument encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded. The initial default facility is LOG_USER.

The *openlog*( ) and *syslog*( ) functions may allocate a file descriptor. It is not necessary to call *openlog*( ) prior to calling *syslog*( ).

The *closelog*( ) function shall close any open file descriptors allocated by previous calls to *openlog*( ) or *syslog*( ).

The *setlogmask*( ) function shall set the log priority mask for the current process to *maskpri* and return the previous mask. If the *maskpri* argument is 0, the current log mask is not modified. Calls by the current process to *syslog*( ) with a priority not set in *maskpri* shall be rejected. The default log mask allows all priorities to be logged. A call to *openlog*( ) is not required prior to calling *setlogmask*( ).

Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are defined in the **<syslog.h>** header.

RETURN VALUE

The *setlogmask*( ) function shall return the previous log priority mask. The *closelog*( ), *openlog*( ), and *syslog*( ) functions shall not return a value.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ERRORS

No errors are defined.

EXAMPLES

Using openlog()

The following example causes subsequent calls to *syslog*( ) to log the process ID with each message, and to write messages to the system console if they cannot be sent to the logging facility.

```
#include <syslog.h>

char *ident = "Process demo";
int logopt = LOG_PID | LOG_CONS;
int facility = LOG_USER;
...
openlog(ident, logopt, facility);
```

Using setlogmask()

The following example causes subsequent calls to *syslog*( ) to accept error messages, and to reject all other messages.

```
#include <syslog.h>

int result;
int mask = LOG_MASK (LOG_ERR);
...
result = setlogmask(mask);
```

Using syslog

The following example sends the message `"This is a message"` to the default logging facility, marking the message as an error message generated by random processes.

```
#include <syslog.h>

char *message = "This is a message";
int priority = LOG_ERR | LOG_USER;
...
syslog(priority, message);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

printf( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<syslog.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7183 **CHANGE HISTORY**

7184 First released in Issue 4, Version 2.

7185 **Issue 5**

7186 Moved from X/OPEN UNIX extension to BASE.

7187 **Issue 6**

7188 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES |
7189 section. |

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7190 **NAME**

7191 confstr — get configurable variables

7192 **SYNOPSIS**

7193 `#include <unistd.h>`

7194 `size_t confstr(int name, char *buf, size_t len);`

7195 **DESCRIPTION**

7196 The *confstr*( ) function shall return configuration-defined string values. Its use and purpose are
7197 similar to *sysconf*( ), but it is used where string values rather than numeric values are returned.

7198 The *name* argument represents the system variable to be queried. The implementation shall
7199 support the following name values, defined in **<unistd.h>**. It may support others:

7200 _CS_PATH
7201 _CS_POSIX_V6_ILP32_OFF32_CFLAGS
7202 _CS_POSIX_V6_ILP32_OFF32_LDFLAGS
7203 _CS_POSIX_V6_ILP32_OFF32_LIBS
7204 _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
7205 _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
7206 _CS_POSIX_V6_ILP32_OFFBIG_LIBS
7207 _CS_POSIX_V6_LP64_OFF64_CFLAGS
7208 _CS_POSIX_V6_LP64_OFF64_LDFLAGS
7209 _CS_POSIX_V6_LP64_OFF64_LIBS
7210 _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
7211 _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
7212 _CS_POSIX_V6_LPBIG_OFFBIG_LIBS
7213 _CS_POSIX_V6_WIDTH_RESTRICTED_ENVS
7214 XSI _CS_XBS5_ILP32_OFF32_CFLAGS (**LEGACY**)
7215 _CS_XBS5_ILP32_OFF32_LDFLAGS (**LEGACY**)
7216 _CS_XBS5_ILP32_OFF32_LIBS (**LEGACY**)
7217 _CS_XBS5_ILP32_OFF32_LINTFLAGS (**LEGACY**)
7218 _CS_XBS5_ILP32_OFFBIG_CFLAGS (**LEGACY**)
7219 _CS_XBS5_ILP32_OFFBIG_LDFLAGS (**LEGACY**)
7220 _CS_XBS5_ILP32_OFFBIG_LIBS (**LEGACY**)
7221 _CS_XBS5_ILP32_OFFBIG_LINTFLAGS (**LEGACY**)
7222 _CS_XBS5_LP64_OFF64_CFLAGS (**LEGACY**)
7223 _CS_XBS5_LP64_OFF64_LDFLAGS (**LEGACY**)
7224 _CS_XBS5_LP64_OFF64_LIBS (**LEGACY**)
7225 _CS_XBS5_LP64_OFF64_LINTFLAGS (**LEGACY**)
7226 _CS_XBS5_LPBIG_OFFBIG_CFLAGS (**LEGACY**)
7227 _CS_XBS5_LPBIG_OFFBIG_LDFLAGS (**LEGACY**)
7228 _CS_XBS5_LPBIG_OFFBIG_LIBS (**LEGACY**)
7229 _CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (**LEGACY**)
7230

7231 If *len* is not 0, and if *name* has a configuration-defined value, *confstr*( ) shall copy that value into
7232 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,
7233 including the terminating null, then *confstr*( ) shall truncate the string to *len*–1 bytes and null-
7234 terminate the result. The application can detect that the string was truncated by comparing the
7235 value returned by *confstr*( ) with *len*.

7236 If *len* is 0 and *buf* is a null pointer, then *confstr*( ) shall still return the integer value as defined
7237 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is
7238 unspecified.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

```
confstr(_CS_PATH, buf, sizeof(buf))
```

can be used as a value of the *PATH* environment variable that accesses all of the standard utilities of IEEE Std 1003.1-2001, if the return value is less than or equal to *sizeof*(*buf*).

RETURN VALUE

If *name* has a configuration-defined value, *confstr*( ) shall return the size of buffer that would be needed to hold the entire configuration-defined value including the terminating null. If this return value is greater than *len*, the string returned in *buf* is truncated.

If *name* is invalid, *confstr*( ) shall return 0 and set *errno* to indicate the error.

If *name* does not have a configuration-defined value, *confstr*( ) shall return 0 and leave *errno* unchanged.

ERRORS

The *confstr*( ) function shall fail if:

[EINVAL] The value of the *name* argument is invalid.

EXAMPLES

None.

APPLICATION USAGE

An application can distinguish between an invalid *name* parameter value and one that corresponds to a configurable variable that has no configuration-defined value by checking if *errno* is modified. This mirrors the behavior of *sysconf*( ).

The original need for this function was to provide a way of finding the configuration-defined default value for the environment variable *PATH*. Since *PATH* can be modified by the user to include directories that could contain utilities replacing the standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001, applications need a way to determine the system-supplied *PATH* environment variable value that contains the correct search path for the standard utilities.

An application could use:

```
confstr(name, (char *)NULL, (size_t)0)
```

to find out how big a buffer is needed for the string value; use *malloc*( ) to allocate a buffer to hold the string; and call *confstr*( ) again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most answers (perhaps 512 or 1 024 bytes), but then use *malloc*( ) to allocate a larger buffer if it finds that this is too small.

RATIONALE

Application developers can normally determine any configuration variable by means of reading from the stream opened by a call to:

```
popen("command -p getconf variable", "r");
```

The *confstr*( ) function with a *name* argument of _CS_PATH returns a string that can be used as a *PATH* environment variable setting that will reference the standard shell and utilities as described in the Shell and Utilities volume of IEEE Std 1003.1-2001.

The *confstr*( ) function copies the returned string into a buffer supplied by the application instead of returning a pointer to a string. This allows a cleaner function in some implementations (such as those with lightweight threads) and resolves questions about when the application must copy the string returned.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

pathconf( ), *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, *c99*

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

A table indicating the permissible values of *name* is added to the DESCRIPTION. All those marked EX are new in this issue.

Issue 6

The Open Group Corrigendum U033/7 is applied. The return value for the case returning the size of the buffer now explicitly states that this includes the terminating null.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated with new arguments which can be used to determine configuration strings for C compiler flags, linker/loader flags, and libraries for each different supported programming environment. This is a change to support data size neutrality.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION is updated to include text describing how _CS_PATH can be used to obtain a *PATH* to access the standard utilities.

The macros associated with the *c89* programming models are marked LEGACY and new equivalent macros associated with *c99* are introduced.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

conj, conjf, conjl — complex conjugate functions

SYNOPSIS

```
#include <complex.h>

double complex conj(double complex z);
float complex conjf(float complex z);
long double complex conjl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary part.

RETURN VALUE

These functions return the complex conjugate value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *cproj*( ), *creal*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7335 **NAME**

7336 connect — connect a socket

7337 **SYNOPSIS**

7338 `#include <sys/socket.h>`

7339 `int connect(int socket, const struct sockaddr *address,`
7340 `socklen_t address_len);`

7341 **DESCRIPTION**

7342 The *connect*( ) function shall attempt to make a connection on a socket. The function takes the
7343 following arguments:

7344 *socket* Specifies the file descriptor associated with the socket.

7345 *address* Points to a **sockaddr** structure containing the peer address. The length and
7346 format of the address depend on the address family of the socket.

7347 *address_len* Specifies the length of the **sockaddr** structure pointed to by the *address*
7348 argument.

7349 If the socket has not already been bound to a local address, *connect*( ) shall bind it to an address
7350 which, unless the socket's address family is AF_UNIX, is an unused local address.

7351 If the initiating socket is not connection-mode, then *connect*( ) shall set the socket's peer address,
7352 and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all
7353 datagrams are sent on subsequent *send*( ) functions, and limits the remote sender for subsequent
7354 *recv*( ) functions. If *address* is a null address for the protocol, the socket's peer address shall be
7355 reset.

7356 If the initiating socket is connection-mode, then *connect*( ) shall attempt to establish a connection
7357 to the address specified by the *address* argument. If the connection cannot be established
7358 immediately and O_NONBLOCK is not set for the file descriptor for the socket, *connect*( ) shall
7359 block for up to an unspecified timeout interval until the connection is established. If the timeout
7360 interval expires before the connection is established, *connect*( ) shall fail and the connection
7361 attempt shall be aborted. If *connect*( ) is interrupted by a signal that is caught while blocked
7362 waiting to establish a connection, *connect*( ) shall fail and set *errno* to [EINTR], but the connection
7363 request shall not be aborted, and the connection shall be established asynchronously.

7364 If the connection cannot be established immediately and O_NONBLOCK is set for the file
7365 descriptor for the socket, *connect*( ) shall fail and set *errno* to [EINPROGRESS], but the connection
7366 request shall not be aborted, and the connection shall be established asynchronously.
7367 Subsequent calls to *connect*( ) for the same socket, before the connection is established, shall fail
7368 and set *errno* to [EALREADY].

7369 When the connection has been established asynchronously, *select*( ) and *poll*( ) shall indicate that
7370 the file descriptor for the socket is ready for writing.

7371 The socket in use may require the process to have appropriate privileges to use the *connect*( )
7372 function.

7373 **RETURN VALUE**

7374 Upon successful completion, *connect*( ) shall return 0; otherwise, –1 shall be returned and *errno*
7375 set to indicate the error.

7376 **ERRORS**

7377 The *connect*( ) function shall fail if:

7378 [EADDRNOTAVAIL]
7379 The specified address is not available from the local machine.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7380 [EAFNOSUPPORT]
7381 The specified address is not a valid address for the address family of the
7382 specified socket.

7383 [EALREADY] A connection request is already in progress for the specified socket.

7384 [EBADF] The *socket* argument is not a valid file descriptor.

7385 [ECONNREFUSED]
7386 The target address was not listening for connections or refused the connection
7387 request.

7388 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the connection
7389 cannot be immediately established; the connection shall be established
7390 asynchronously.

7391 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal
7392 that was caught; the connection shall be established asynchronously.

7393 [EISCONN] The specified socket is connection-mode and is already connected.

7394 [ENETUNREACH]
7395 No route to the network is present.

7396 [ENOTSOCK] The *socket* argument does not refer to a socket.

7397 [EPROTOTYPE] The specified address has a different type than the socket bound to the
7398 specified peer address.

7399 [ETIMEDOUT] The attempt to connect timed out before a connection was made.

7400 If the address family of the socket is AF_UNIX, then *connect*( ) shall fail if:

7401 [EIO] An I/O error occurred while reading from or writing to the file system.

7402 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
7403 in *address*.

7404 [ENAMETOOLONG]
7405 A component of a pathname exceeded {NAME_MAX} characters, or an entire
7406 pathname exceeded {PATH_MAX} characters.

7407 [ENOENT] A component of the pathname does not name an existing file or the pathname
7408 is an empty string.

7409 [ENOTDIR] A component of the path prefix of the pathname in *address* is not a directory.

7410 The *connect*( ) function may fail if:

7411 [EACCES] Search permission is denied for a component of the path prefix; or write
7412 access to the named socket is denied.

7413 [EADDRINUSE] Attempt to establish a connection that uses addresses that are already in use.

7414 [ECONNRESET] Remote host reset the connection request.

7415 [EHOSTUNREACH]
7416 The destination host cannot be reached (probably because the host is down or
7417 a remote router cannot reach it).

7418 [EINVAL] The *address_len* argument is not a valid length for the address family; or
7419 invalid address family in the **sockaddr** structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7420 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
7421 resolution of the pathname in *address*.

7422 [ENAMETOOLONG]
7423 Pathname resolution of a symbolic link produced an intermediate result
7424 whose length exceeds {PATH_MAX}.

7425 [ENETDOWN] The local network interface used to reach the destination is down.

7426 [ENOBUFS] No buffer space is available.

7427 [EOPNOTSUPP] The socket is listening and cannot be connected.

7428 **EXAMPLES**

7429 None.

7430 **APPLICATION USAGE**

7431 If *connect*( ) fails, the state of the socket is unspecified. Conforming applications should close the
7432 file descriptor and create a new socket before attempting to reconnect.

7433 **RATIONALE**

7434 None.

7435 **FUTURE DIRECTIONS**

7436 None.

7437 **SEE ALSO**

7438 *accept*( ), *bind*( ), *close*( ), *getsockname*( ), *poll*( ), *select*( ), *send*( ), *shutdown*( ), *socket*( ), the Base
7439 Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

7440 **CHANGE HISTORY**

7441 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7442 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
7443 [ELOOP] error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

copysign, copysignf, copysignl — number manipulation function

SYNOPSIS

```
#include <math.h>

double copysign(double x, double y);
float copysignf(float x, float y);
long double copysignl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall produce a value with the magnitude of *x* and the sign of *y*. On implementations that represent a signed zero but do not treat negative zero consistently in arithmetic operations, these functions regard the sign of zero as positive.

RETURN VALUE

Upon successful completion, these functions shall return a value with the magnitude of *x* and the sign of *y*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

signbit( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

cos, cosf, cosl — cosine function

SYNOPSIS

```
#include <math.h>

double cos(double x);
float cosf(float x);
long double cosl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the cosine of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the cosine of *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, the value 1.0 shall be returned.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is ±Inf.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

Taking the Cosine of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45 * M_PI / 180;
double result;
...
result = cos(radians);
```

APPLICATION USAGE

These functions may lose accuracy when their argument is near an odd multiple of π/2 or is far from 0.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7517 **RATIONALE**

7518 None.

7519 **FUTURE DIRECTIONS**

7520 None.

7521 **SEE ALSO**

7522 *acos*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sin*( ), *tan*( ), the Base Definitions volume of
7523 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
7524 **<math.h>**

7525 **CHANGE HISTORY**

7526 First released in Issue 1. Derived from Issue 1 of the SVID.

7527 **Issue 5**

7528 The DESCRIPTION is updated to indicate how an application should check for an error. This
7529 text was previously published in the APPLICATION USAGE section.

7530 **Issue 6**

7531 The *cosf*( ) and *cosl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

7532 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
7533 revised to align with the ISO / IEC 9899: 1999 standard.

7534 IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are
7535 marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

cosh, coshf, coshl — hyperbolic cosine functions

SYNOPSIS

```
#include <math.h>

double cosh(double x);
float coshf(float x);
long double coshl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the hyperbolic cosine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic cosine of *x*.

If the correct value would cause overflow, a range error shall occur and *cosh*( ), *coshf*( ), and *coshl*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, the value 1.0 shall be returned.

If *x* is ±Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

Range Error — The result would cause an overflow.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

For IEEE Std 754-1985 **double**, 710.5 < |*x*| implies that *cosh*(*x*) has overflowed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE ALSO

acosh( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sinh*( ), *tanh*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *coshf*( ) and *coshl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7592 **NAME**

7593 cosl — cosine function

7594 **SYNOPSIS**

7595
```
#include <math.h>
```

7596
```
long double cosl(long double x);
```

7597 **DESCRIPTION**

7598 Refer to *cos*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

7599 **NAME**

7600 cpow, cpowf, cpowl — complex power functions

7601 **SYNOPSIS**

7602
```
#include <complex.h>
```

7603
```
double complex cpow(double complex x, double complex y);
```
7604
```
float complex cpowf(float complex x, float complex y);
```
7605
```
long double complex cpowl(long double complex x,
```
7606
```
    long double complex y);
```

7607 **DESCRIPTION**

7608 CX The functionality described on this reference page is aligned with the ISO C standard. Any
7609 conflict between the requirements described here and the ISO C standard is unintentional. This
7610 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7611 These functions shall compute the complex power function x^y, with a branch cut for the first
7612 parameter along the negative real axis.

7613 **RETURN VALUE**

7614 These functions shall return the complex power function value.

7615 **ERRORS**

7616 No errors are defined.

7617 **EXAMPLES**

7618 None.

7619 **APPLICATION USAGE**

7620 None.

7621 **RATIONALE**

7622 None.

7623 **FUTURE DIRECTIONS**

7624 None.

7625 **SEE ALSO**

7626 *cabs*( ), *csqrt*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

7627 **CHANGE HISTORY**

7628 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7629 **NAME**

7630 cproj, cprojf, cprojl — complex projection functions

7631 **SYNOPSIS**

7632
```
#include <complex.h>

double complex cproj(double complex z);
float complex cprojf(float complex z);
long double complex cprojl(long double complex z);
```

7636 **DESCRIPTION**

7637 CX The functionality described on this reference page is aligned with the ISO C standard. Any
7638 conflict between the requirements described here and the ISO C standard is unintentional. This
7639 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7640 These functions shall compute a projection of *z* onto the Riemann sphere: *z* projects to *z*, except
7641 that all complex infinities (even those with one infinite part and one NaN part) project to
7642 positive infinity on the real axis. If *z* has an infinite part, then *cproj*(*z*) shall be equivalent to:

7643
```
INFINITY + I * copysign(0.0, cimag(z))
```

7644 **RETURN VALUE**

7645 These functions shall return the value of the projection onto the Riemann sphere.

7646 **ERRORS**

7647 No errors are defined.

7648 **EXAMPLES**

7649 None.

7650 **APPLICATION USAGE**

7651 None.

7652 **RATIONALE**

7653 Two topologies are commonly used in complex mathematics: the complex plane with its
7654 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is
7655 better suited for transcendental functions, the Riemann sphere for algebraic functions. The
7656 complex types with their multiplicity of infinities provide a useful (though imperfect) model for
7657 the complex plane. The *cproj*( ) function helps model the Riemann sphere by mapping all
7658 infinities to one, and should be used just before any operation, especially comparisons, that
7659 might give spurious results for any of the other infinities. Note that a complex value with one
7660 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is
7661 infinite, the complex value is infinite independent of the value of the other part. For the same
7662 reason, *cabs*( ) returns an infinity if its argument has an infinite part and a NaN part.

7663 **FUTURE DIRECTIONS**

7664 None.

7665 **SEE ALSO**

7666 *carg*( ), *cimag*( ), *conj*( ), *creal*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

7667 **CHANGE HISTORY**

7668 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

creal, crealf, creall — complex real functions

SYNOPSIS

```
#include <complex.h>

double creal(double complex z);
float crealf(float complex z);
long double creall(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the real part of *z*.

RETURN VALUE

These functions shall return the real part value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

For a variable *z* of type **complex**:

```
z == creal(z) + cimag(z)*I
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *conj*( ), *cproj*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

creat — create a new file or rewrite an existing one

SYNOPSIS

OH
```
#include <sys/stat.h>
#include <fcntl.h>

int creat(const char *path, mode_t mode);
```

DESCRIPTION

The function call:

```
creat(path, mode)
```

shall be equivalent to:

```
open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)
```

RETURN VALUE

Refer to *open*( ).

ERRORS

Refer to *open*( ).

EXAMPLES

Creating a File

The following example creates the file **/tmp/file** with read and write permissions for the file owner and read permission for group and others. The resulting file descriptor is assigned to the *fd* variable.

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *filename = "/tmp/file";
...
fd = creat(filename, mode);
...
```

APPLICATION USAGE

None.

RATIONALE

The *creat*( ) function is redundant. Its services are also provided by the *open*( ) function. It has been included primarily for historical purposes since many existing applications depend on it. It is best considered a part of the C binding rather than a function that should be provided in other languages.

FUTURE DIRECTIONS

None.

SEE ALSO

open( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7739 **CHANGE HISTORY**

7740 First released in Issue 1. Derived from Issue 1 of the SVID.

7741 **Issue 6**

7742 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

7743 The following new requirements on POSIX implementations derive from alignment with the
7744 Single UNIX Specification:

7745 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
7746 required for conforming implementations of previous POSIX specifications, it was not
7747 required for UNIX applications.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

crypt — string encoding function (**CRYPT**)

SYNOPSIS

XSI

```
#include <unistd.h>

char *crypt(const char *key, const char *salt);
```

DESCRIPTION

The *crypt*( ) function is a string encoding function. The algorithm is implementation-defined.

The *key* argument points to a string to be encoded. The *salt* argument is a string chosen from the set:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 . /
```

The first two characters of this string may be used to perturb the encoding algorithm.

The return value of *crypt*( ) points to static data that is overwritten by each call.

The *crypt*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *crypt*( ) shall return a pointer to the encoded string. The first two characters of the returned value shall be those of the *salt* argument. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *crypt*( ) function shall fail if:

[ENOSYS] The functionality is not supported on this implementation.

EXAMPLES

Encoding Passwords

The following example finds a user database entry matching a particular user name and changes the current password to a new password. The *crypt*( ) function generates an encoded version of each password. The first call to *crypt*( ) produces an encoded version of the old password; that encoded password is then compared to the password stored in the user database. The second call to *crypt*( ) encodes the new password before it is stored.

The *putpwent*( ) function, used in the following example, is not part of IEEE Std 1003.1-2001.

```
#include <unistd.h>
#include <pwd.h>
#include <string.h>
#include <stdio.h>
...
int valid_change;
int pfd;  /* Integer for file descriptor returned by open(). */
FILE *fpfd;  /* File pointer for use in putpwent(). */
struct passwd *p;
char user[100];
char oldpasswd[100];
char newpasswd[100];
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        char savepasswd[100];
        ...
        valid_change = 0;
        while ((p = getpwent()) != NULL) {
            /* Change entry if found. */
            if (strcmp(p->pw_name, user) == 0) {
                if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
                    strcpy(savepasswd, crypt(newpasswd, user));
                    p->pw_passwd = savepasswd;
                    valid_change = 1;
                }
                else {
                    fprintf(stderr, "Old password is not valid\n");
                }
            }
            /* Put passwd entry into ptmp. */
            putpwent(p, fpfd);
        }
```

APPLICATION USAGE

The values returned by this function need not be portable among XSI-conformant systems.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

encrypt( ), *setkey*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7823 **NAME**

7824 csin, csinf, csinl — complex sine functions

7825 **SYNOPSIS**

7826 ```
#include <complex.h>
```

7827 ```
double complex csin(double complex z);
```
7828 ```
float complex csinf(float complex z);
```
7829 ```
long double complex csinl(long double complex z);
```

7830 **DESCRIPTION**

7831 CX The functionality described on this reference page is aligned with the ISO C standard. Any
7832 conflict between the requirements described here and the ISO C standard is unintentional. This
7833 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7834 These functions shall compute the complex sine of *z*.

7835 **RETURN VALUE**

7836 These functions shall return the complex sine value.

7837 **ERRORS**

7838 No errors are defined.

7839 **EXAMPLES**

7840 None.

7841 **APPLICATION USAGE**

7842 None.

7843 **RATIONALE**

7844 None.

7845 **FUTURE DIRECTIONS**

7846 None.

7847 **SEE ALSO**

7848 *casin*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

7849 **CHANGE HISTORY**

7850 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

csinh, csinhf, csinhl — complex hyperbolic sine functions

SYNOPSIS

```
#include <complex.h>

double complex csinh(double complex z);
float complex csinhf(float complex z);
long double complex csinhl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex hyperbolic sine of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic sine value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

casinh( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7879 **NAME**

7880 csinl — complex sine functions

7881 **SYNOPSIS**

7882 `#include <complex.h>`

7883 `long double complex csinl(long double complex z);`

7884 **DESCRIPTION**

7885 Refer to *csin*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

csqrt, csqrtf, csqrtl — complex square root functions

SYNOPSIS

```
#include <complex.h>

double complex csqrt(double complex z);
float complex csqrtf(float complex z);
long double complex csqrtl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex square root of *z*, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the complex square root value, in the range of the right half-plane (including the imaginary axis).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cabs( ), *cpow*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

ctan, ctanf, ctanl — complex tangent functions

SYNOPSIS

```
#include <complex.h>

double complex ctan(double complex z);
float complex ctanf(float complex z);
long double complex ctanl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex tangent of *z*.

RETURN VALUE

These functions shall return the complex tangent value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catan( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

SYNOPSIS

```
#include <complex.h>

double complex ctanh(double complex z);
float complex ctanhf(float complex z);
long double complex ctanhl(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complex hyperbolic tangent of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic tangent value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catanh( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7972 **NAME**

7973 ctanl — complex tangent functions

7974 **SYNOPSIS**

7975
```
#include <complex.h>
```

7976
```
long double complex ctanl(long double complex z);
```

7977 **DESCRIPTION**

7978 Refer to *ctan*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

ctermid — generate a pathname for the controlling terminal

SYNOPSIS

CX
```
#include <stdio.h>

char *ctermid(char *s);
```

DESCRIPTION

The *ctermid*( ) function shall generate a string that, when used as a pathname, refers to the current controlling terminal for the current process. If *ctermid*( ) returns a pathname, access to the file is not guaranteed.

If the application uses any of the _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS functions, it shall ensure that the *ctermid*( ) function is called with a non-NULL parameter.

RETURN VALUE

If *s* is a null pointer, the string shall be generated in an area that may be static (and therefore may be overwritten by each call), the address of which shall be returned. Otherwise, *s* is assumed to point to a character array of at least L_ctermid bytes; the string is placed in this array and the value of *s* shall be returned. The symbolic constant L_ctermid is defined in **<stdio.h>**, and shall have a value greater than 0.

The *ctermid*( ) function shall return an empty string if the pathname that would refer to the controlling terminal cannot be determined, or if the function is unsuccessful.

ERRORS

No errors are defined.

EXAMPLES

Determining the Controlling Terminal for the Current Process

The following example returns a pointer to a string that identifies the controlling terminal for the current process. The pathname for the terminal is stored in the array pointed to by the *ptr* argument, which has a size of L_ctermid bytes, as indicated by the *term* argument.

```
#include <stdio.h>
...
char term[L_ctermid];
char *ptr;

ptr = ctermid(term);
```

APPLICATION USAGE

The difference between *ctermid*( ) and *ttyname*( ) is that *ttyname*( ) must be handed a file descriptor and return a path of the terminal associated with that file descriptor, while *ctermid*( ) returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a pathname.

RATIONALE

L_ctermid must be defined appropriately for a given implementation and must be greater than zero so that array declarations using it are accepted by the compiler. The value includes the terminating null byte.

Conforming applications that use threads cannot call *ctermid*( ) with NULL as the parameter if either _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS is defined. If *s* is not NULL, the *ctermid*( ) function generates a string that, when used as a pathname, refers to the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

current controlling terminal for the current process. If *s* is NULL, the return value of *ctermid*( ) is undefined.

There is no additional burden on the programmer—changing to use a hypothetical thread-safe version of *ctermid*( ) along with allocating a buffer is more of a burden than merely allocating a buffer. Application code should not assume that the returned string is short, as some implementations have more than two pathname components before reaching a logical device name.

FUTURE DIRECTIONS

None.

SEE ALSO

ttyname( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ctime, ctime_r — convert a time value to a date and time string

SYNOPSIS

```
#include <time.h>

char *ctime(const time_t *clock);
TSF char *ctime_r(const time_t *clock, char *buf);
```

DESCRIPTION

CX For *ctime*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *ctime*() function shall convert the time pointed to by *clock*, representing time in seconds since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
asctime(localtime(clock))
```

CX The *asctime*(), *ctime*(), *gmtime*(), and *localtime*() functions shall return values in one of two static objects: a broken-down time structure and an array of **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The *ctime*() function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

TSF The *ctime_r*() function shall convert the calendar time pointed to by *clock* to local time in exactly the same form as *ctime*() and put the string into the array pointed to by *buf* (which shall be at least 26 bytes in size) and return *buf*.

Unlike *ctime*(), the thread-safe version *ctime_r*() is not required to set *tzname*.

RETURN VALUE

The *ctime*() function shall return the pointer returned by *asctime*() with that broken-down time as an argument.

TSF Upon successful completion, *ctime_r*() shall return a pointer to the string pointed to by *buf*. When an error is encountered, a null pointer shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Values for the broken-down time structure can be obtained by calling *gmtime*() or *localtime*(). The *ctime*() function is included for compatibility with older implementations, and does not support localized date and time formats. Applications should use the *strftime*() function to achieve maximum portability.

The *ctime_r*() function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8081 **FUTURE DIRECTIONS**

8082 None.

8083 **SEE ALSO**

8084 *asctime*( ), *clock*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( ),
8085 the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

8086 **CHANGE HISTORY**

8087 First released in Issue 1. Derived from Issue 1 of the SVID.

8088 **Issue 5**

8089 Normative text previously in the APPLICATION USAGE section is moved to the
8090 DESCRIPTION.

8091 The *ctime_r*( ) function is included for alignment with the POSIX Threads Extension.

8092 A note indicating that the *ctime*( ) function need not be reentrant is added to the DESCRIPTION.

8093 **Issue 6**

8094 Extensions beyond the ISO C standard are marked.

8095 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8096 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
8097 its avoidance of possibly using a static data area.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8098 **NAME**

8099 daylight — daylight savings time flag

8100 **SYNOPSIS**

8101 XSI
```
#include <time.h>
```

8102
```
extern int daylight;
```

8103

8104 **DESCRIPTION**

8105 Refer to *tzset*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store — database functions

SYNOPSIS

XSI

```
#include <ndbm.h>

int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

DESCRIPTION

These functions create, access, and modify a database.

A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in the object pointed to by *dptr*.

The database is stored in two files. One file is a directory containing a bitmap of keys and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

The *dbm_open*( ) function shall open a database. The *file* argument to the function is the pathname of the database. The function opens two files named *file***.dir** and *file***.pag**. The *open_flags* argument has the same meaning as the *flags* argument of *open*( ) except that a database opened for write-only access opens the files for read and write access and the behavior of the O_APPEND flag is unspecified. The *file_mode* argument has the same meaning as the third argument of *open*( ).

The *dbm_close*( ) function shall close a database. The application shall ensure that argument *db* is a pointer to a **dbm** structure that has been returned from a call to *dbm_open*( ).

These database functions shall support an internal block size large enough to support key/content pairs of at least 1 023 bytes.

The *dbm_fetch*( ) function shall read a record from a database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The argument *key* is a **datum** that has been initialized by the application to the value of the key that matches the key of the record the program is fetching.

The *dbm_store*( ) function shall write a record to a database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The argument *key* is a **datum** that has been initialized by the application to the value of the key that identifies (for subsequent reading, writing, or deleting) the record the application is writing. The argument *content* is a **datum** that has been initialized by the application to the value of the record the program is writing. The argument *store_mode* controls whether *dbm_store*( ) replaces any pre-existing record that has the same key that is specified by the *key* argument. The application shall set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be replaced with the new record. If the database contains a record that matches the *key* argument and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ignored. If the database does not contain a record that matches the *key* argument and *store_mode* is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

If the sum of a key/content pair exceeds the internal block size, the result is unspecified. Moreover, the application shall ensure that all key/content pairs that hash together fit on a single block. The *dbm_store*( ) function shall return an error in the event that a disk block fills with inseparable data.

The *dbm_delete*( ) function shall delete a record and its key from the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The argument *key* is a **datum** that has been initialized by the application to the value of the key that identifies the record the program is deleting.

The *dbm_firstkey*( ) function shall return the first key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ).

The *dbm_nextkey*( ) function shall return the next key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ). The application shall ensure that the *dbm_firstkey*( ) function is called before calling *dbm_nextkey*( ). Subsequent calls to *dbm_nextkey*( ) return the next key until all of the keys in the database have been returned.

The *dbm_error*( ) function shall return the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ).

The *dbm_clearerr*( ) function shall clear the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*( ).

The *dptr* pointers returned by these functions may point into static storage that may be changed by subsequent calls.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *dbm_store*( ) and *dbm_delete*( ) functions shall return 0 when they succeed and a negative value when they fail.

The *dbm_store*( ) function shall return 1 if it is called with a *flags* value of DBM_INSERT and the function finds an existing record with the same key.

The *dbm_error*( ) function shall return 0 if the error condition is not set and return a non-zero value if the error condition is set.

The return value of *dbm_clearerr*( ) is unspecified.

The *dbm_firstkey*( ) and *dbm_nextkey*( ) functions shall return a key **datum**. When the end of the database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr* member of the key shall be a null pointer and the error condition of the database shall be set.

The *dbm_fetch*( ) function shall return a content **datum**. If no record in the database matches the key or if an error condition has been detected in the database, the *dptr* member of the content shall be a null pointer.

The *dbm_open*( ) function shall return a pointer to a database structure. If an error is detected during the operation, *dbm_open*( ) shall return a (**DBM ***)0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The following code can be used to traverse the database:

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

The *dbm_** functions provided in this library should not be confused in any way with those of a general-purpose database management system. These functions do not provide for multiple search keys per entry, they do not protect against multi-user access (in other words they do not lock records or files), and they do not provide the many other useful database functions that are found in more robust database management systems. Creating and updating databases by use of these functions is relatively slow because of data copies that occur upon hash collisions. These functions are useful for applications requiring fast lookup of relatively static information that is to be indexed by a single key.

Note that a strictly conforming application is extremely limited by these functions: since there is no way to determine that the keys in use do not all hash to the same value (although that would be rare), a strictly conforming application cannot be guaranteed that it can store more than one block's worth of data in the database. As long as a key collision does not occur, additional data may be stored, but because there is no way to determine whether an error is due to a key collision or some other error condition (*dbm_error*( ) being effectively a Boolean), once an error is detected, the application is effectively limited to guessing what the error might be if it wishes to continue using these functions.

The *dbm_delete*( ) function need not physically reclaim file space, although it does make it available for reuse by the database.

After calling *dbm_store*( ) or *dbm_delete*( ) during a pass through the keys by *dbm_firstkey*( ) and *dbm_nextkey*( ), the application should reset the database by calling *dbm_firstkey*( ) before again calling *dbm_nextkey*( ). The contents of these files are unspecified and may not be portable.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

open( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<ndbm.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8235 **Issue 6**

8236 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8237 **NAME**

8238 difftime — compute the difference between two calendar time values

8239 **SYNOPSIS**

8240 ```
#include <time.h>
```

8241 ```
double difftime(time_t time1, time_t time0);
```

8242 **DESCRIPTION**

8243 CX The functionality described on this reference page is aligned with the ISO C standard. Any
8244 conflict between the requirements described here and the ISO C standard is unintentional. This
8245 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8246 The *difftime*( ) function shall compute the difference between two calendar times (as returned by
8247 *time*( )): *time*1– *time*0.

8248 **RETURN VALUE**

8249 The *difftime*( ) function shall return the difference expressed in seconds as a type **double**.

8250 **ERRORS**

8251 No errors are defined.

8252 **EXAMPLES**

8253 None.

8254 **APPLICATION USAGE**

8255 None.

8256 **RATIONALE**

8257 None.

8258 **FUTURE DIRECTIONS**

8259 None.

8260 **SEE ALSO**

8261 *asctime*( ), *clock*( ), *ctime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( ),
8262 the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

8263 **CHANGE HISTORY**

8264 First released in Issue 4. Derived from the ISO C standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

dirname — report the parent directory name of a file pathname

SYNOPSIS

XSI
```
#include <libgen.h>

char *dirname(char *path);
```

DESCRIPTION

The *dirname*( ) function shall take a pointer to a character string that contains a pathname, and return a pointer to a string that is a pathname of the parent directory of that file. Trailing '/' characters in the path are not counted as part of the path.

If *path* does not contain a '/', then *dirname*( ) shall return a pointer to the string ".". If *path* is a null pointer or points to an empty string, *dirname*( ) shall return a pointer to the string ".".

The *dirname*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *dirname*( ) function shall return a pointer to a string that is the parent directory of *path*. If *path* is a null pointer or points to an empty string, a pointer to a string "." is returned.

The *dirname*( ) function may modify the string pointed to by *path*, and may return a pointer to static storage that may then be overwritten by subsequent calls to *dirname*( ).

ERRORS

No errors are defined.

EXAMPLES

The following code fragment reads a pathname, changes the current working directory to the parent directory, and opens the file.

```
char path[PATH_MAX], *pathcopy;
int fd;
fgets(path, PATH_MAX, stdin);
pathcopy = strdup(path);
chdir(dirname(pathcopy));
fd = open(basename(path), O_RDONLY);
```

Sample Input and Output Strings for dirname( )

In the following table, the input string is the value pointed to by *path*, and the output string is the return value of the *dirname*( ) function.

Input String	Output String
"/usr/lib"	"/usr"
"/usr/"	"/"
"usr"	"."
"/"	"/"
"."	"."
".."	"."

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Changing the Current Directory to the Parent Directory

The following program fragment reads a pathname, changes the current working directory to the parent directory, and opens the file.

```
#include <unistd.h>
#include <limits.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <libgen.h>
...
char path[PATH_MAX], *pathcopy;
int fd;
...
fgets(path, PATH_MAX, stdin);
pathcopy = strdup(path);
chdir(dirname(pathcopy));
fd = open(basename(path), O_RDONLY);
```

APPLICATION USAGE

The *dirname*( ) and *basename*( ) functions together yield a complete pathname. The expression *dirname*(*path*) obtains the pathname of the directory where *basename*(*path*) is found.

Since the meaning of the leading `"//"` is implementation-defined, *dirname*("**//foo**) may return either `"//"` or `'//'` (but nothing else).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

basename( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<libgen.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

div — compute the quotient and remainder of an integer division

SYNOPSIS

```
#include <stdlib.h>

div_t div(int numer, int denom);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *div*( ) function shall compute the quotient and remainder of the division of the numerator *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, *quot***denom*+*rem* shall equal *numer*.

RETURN VALUE

The *div*( ) function shall return a structure of type **div_t**, comprising both the quotient and the remainder. The structure includes the following members, in any order:

```
int  quot;  /* quotient */
int  rem;   /* remainder */
```

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ldiv( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8372 **NAME**

8373 dlclose — close a *dlopen*( ) object

8374 **SYNOPSIS**

8375 XSI
```
#include <dlfcn.h>
```

8376
```
int dlclose(void *handle);
```

8377

8378 **DESCRIPTION**

8379 The *dlclose*( ) function shall inform the system that the object referenced by a *handle* returned
8380 from a previous *dlopen*( ) invocation is no longer needed by the application.

8381 The use of *dlclose*( ) reflects a statement of intent on the part of the process, but does not create
8382 any requirement upon the implementation, such as removal of the code or symbols referenced
8383 by *handle*. Once an object has been closed using *dlclose*( ) an application should assume that its
8384 symbols are no longer available to *dlsym*( ). All objects loaded automatically as a result of
8385 invoking *dlopen*( ) on the referenced object shall also be closed if this is the last reference to it.

8386 Although a *dlclose*( ) operation is not required to remove structures from an address space,
8387 neither is an implementation prohibited from doing so. The only restriction on such a removal is
8388 that no object shall be removed to which references have been relocated, until or unless all such
8389 references are removed. For instance, an object that had been loaded with a *dlopen*( ) operation
8390 specifying the RTLD_GLOBAL flag might provide a target for dynamic relocations performed in
8391 the processing of other objects—in such environments, an application may assume that no
8392 relocation, once made, shall be undone or remade unless the object requiring the relocation has
8393 itself been removed.

8394 **RETURN VALUE**

8395 If the referenced object was successfully closed, *dlclose*( ) shall return 0. If the object could not be
8396 closed, or if *handle* does not refer to an open object, *dlclose*( ) shall return a non-zero value. More
8397 detailed diagnostic information shall be available through *dlerror*( ).

8398 **ERRORS**

8399 No errors are defined.

8400 **EXAMPLES**

8401 The following example illustrates use of *dlopen*( ) and *dlclose*( ):

```
8402 ...
8403 /* Open a dynamic library and then close it ... */

8404 #include <dlfcn.h>
8405 void *mylib;
8406 int eret;

8407 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
8408 ...
8409 eret = dlclose(mylib);
8410 ...
```

8411 **APPLICATION USAGE**

8412 A conforming application should employ a *handle* returned from a *dlopen*( ) invocation only
8413 within a given scope bracketed by the *dlopen*( ) and *dlclose*( ) operations. Implementations are
8414 free to use reference counting or other techniques such that multiple calls to *dlopen*( ) referencing
8415 the same object may return the same object for *handle*. Implementations are also free to reuse a
8416 *handle*. For these reasons, the value of a *handle* must be treated as an opaque object by the
8417 application, used only in calls to *dlsym*( ) and *dlclose*( ).

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8418 **RATIONALE**

8419 None.

8420 **FUTURE DIRECTIONS**

8421 None.

8422 **SEE ALSO**

8423 *dlerror*( ), *dlopen*( ), *dlsym*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dlfcn.h>**

8424 **CHANGE HISTORY**

8425 First released in Issue 5.

8426 **Issue 6**

8427 The DESCRIPTION is updated to say that the referenced object is closed ‘‘if this is the last
8428 reference to it’’.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

dlerror — get diagnostic information

SYNOPSIS

XSI

```
#include <dlfcn.h>

char *dlerror(void);
```

DESCRIPTION

The *dlerror*( ) function shall return a null-terminated character string (with no trailing <newline>) that describes the last error that occurred during dynamic linking processing. If no dynamic linking errors have occurred since the last invocation of *dlerror*( ), *dlerror*( ) shall return NULL. Thus, invoking *dlerror*( ) a second time, immediately following a prior invocation, shall result in NULL being returned.

The *dlerror*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

If successful, *dlerror*( ) shall return a null-terminated character string; otherwise, NULL shall be returned.

ERRORS

No errors are defined.

EXAMPLES

The following example prints out the last dynamic linking error:

```
...
#include <dlfcn.h>

char *errstr;

errstr = dlerror();
if (errstr != NULL)
printf ("A dynamic linking error occurred: (%s)\n", errstr);
...
```

APPLICATION USAGE

The messages returned by *dlerror*( ) may reside in a static buffer that is overwritten on each call to *dlerror*( ). Application code should not write to this buffer. Programs wishing to preserve an error message should make their own copies of that message. Depending on the application environment with respect to asynchronous execution events, such as signals or other asynchronous computation sharing the address space, conforming applications should use a critical section to retrieve the error pointer and buffer.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose( ), *dlopen*( ), *dlsym*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dlfcn.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8470 **CHANGE HISTORY**

8471 First released in Issue 5.

8472 **Issue 6**

8473 In the DESCRIPTION the note about reentrancy and thread-safety is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

dlopen — gain access to an executable object file

SYNOPSIS

XSI
```
#include <dlfcn.h>

void *dlopen(const char *file, int mode);
```

DESCRIPTION

The *dlopen*( ) function shall make an executable object file specified by *file* available to the calling program. The class of files eligible for this operation and the manner of their construction are implementation-defined, though typically such files are executable objects such as shared libraries, relocatable files, or programs. Note that some implementations permit the construction of dependencies between such objects that are embedded within files. In such cases, a *dlopen*( ) operation shall load such dependencies in addition to the object referenced by *file*. Implementations may also impose specific constraints on the construction of programs that can employ *dlopen*( ) and its related services.

A successful *dlopen*( ) shall return a *handle* which the caller may use on subsequent calls to *dlsym*( ) and *dlclose*( ). The value of this *handle* should not be interpreted in any way by the caller.

The *file* argument is used to construct a pathname to the object file. If *file* contains a slash character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in an implementation-defined manner to yield a pathname.

If the value of *file* is 0, *dlopen*( ) shall provide a *handle* on a global symbol object. This object shall provide access to the symbols from an ordered set of objects consisting of the original program image file, together with any objects loaded at program start-up as specified by that process image file (for example, shared libraries), and the set of objects loaded using a *dlopen*( ) operation together with the RTLD_GLOBAL flag. As the latter set of objects can change during execution, the set identified by *handle* can also change dynamically.

Only a single copy of an object file is brought into the address space, even if *dlopen*( ) is invoked multiple times in reference to the file, and even if different pathnames are used to reference the file.

The *mode* parameter describes how *dlopen*( ) shall operate upon *file* with respect to the processing of relocations and the scope of visibility of the symbols provided within *file*. When an object is brought into the address space of a process, it may contain references to symbols whose addresses are not known until the object is loaded. These references shall be relocated before the symbols can be accessed. The *mode* parameter governs when these relocations take place and may have the following values:

RTLD_LAZY
: Relocations shall be performed at an implementation-defined time, ranging from the time of the *dlopen*( ) call until the first reference to a given symbol occurs. Specifying RTLD_LAZY should improve performance on implementations supporting dynamic symbol binding as a process may not reference all of the functions in any given object. And, for systems supporting dynamic symbol resolution for normal process execution, this behavior mimics the normal handling of process execution.

RTLD_NOW
: All necessary relocations shall be performed when the object is first loaded. This may waste some processing if relocations are performed for functions that are never referenced. This behavior may be useful for applications that need to know as soon as an object is loaded that all

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

symbols referenced during execution are available.

Any object loaded by *dlopen*( ) that requires relocations against global symbols can reference the symbols in the original process image file, any objects loaded at program start-up, from the object itself as well as any other object included in the same *dlopen*( ) invocation, and any objects that were loaded in any *dlopen*( ) invocation and which specified the RTLD_GLOBAL flag. To determine the scope of visibility for the symbols loaded with a *dlopen*( ) invocation, the *mode* parameter should be a bitwise-inclusive OR with one of the following values:

RTLD_GLOBAL — The object's symbols shall be made available for the relocation processing of any other object. In addition, symbol lookup using *dlopen*(*0*, *mode*) and an associated *dlsym*( ) allows objects loaded with this *mode* to be searched.

RTLD_LOCAL — The object's symbols shall not be made available for the relocation processing of any other object.

If neither RTLD_GLOBAL nor RTLD_LOCAL are specified, then an implementation-defined default behavior shall be applied.

If a *file* is specified in multiple *dlopen*( ) invocations, *mode* is interpreted at each invocation. Note, however, that once RTLD_NOW has been specified all relocations shall have been completed rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations irrelevant. Similarly, note that once RTLD_GLOBAL has been specified the object shall maintain the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL, as long as the object remains in the address space (see *dlclose*( )).

Symbols introduced into a program through calls to *dlopen*( ) may be used in relocation activities. Symbols so introduced may duplicate symbols already defined by the program or previous *dlopen*( ) operations. To resolve the ambiguities such a situation might present, the resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two such resolution orders are defined: *load* or *dependency* ordering. Load order establishes an ordering among symbol definitions, such that the definition first loaded (including definitions from the image file and any dependent objects loaded with it) has priority over objects added later (via *dlopen*( )). Load ordering is used in relocation processing. Dependency ordering uses a breadth-first order starting with a given object, then all of its dependencies, then any dependents of those, iterating until all dependencies are satisfied. With the exception of the global symbol object obtained via a *dlopen*( ) operation on a *file* of 0, dependency ordering is used by the *dlsym*( ) function. Load ordering is used in *dlsym*( ) operations upon the global symbol object.

When an object is first made accessible via *dlopen*( ) it and its dependent objects are added in dependency order. Once all the objects are added, relocations are performed using load order. Note that if an object or its dependencies had been previously loaded, the load and dependency orders may yield different resolutions.

The symbols introduced by *dlopen*( ) operations and available through *dlsym*( ) are at a minimum those which are exported as symbols of global scope by the object. Typically such symbols shall be those that were specified in (for example) C source code as having *extern* linkage. The precise manner in which an implementation constructs the set of exported symbols for a *dlopen*( ) object is specified by that implementation.

RETURN VALUE

If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for processing by *dlopen*( ), or if an error occurs during the process of loading *file* or relocating its symbolic references, *dlopen*( ) shall return NULL. More detailed diagnostic information shall be available through *dlerror*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8567 **ERRORS**

8568 No errors are defined.

8569 **EXAMPLES**

8570 None.

8571 **APPLICATION USAGE**

8572 None.

8573 **RATIONALE**

8574 None.

8575 **FUTURE DIRECTIONS**

8576 None.

8577 **SEE ALSO**

8578 *dlclose*( ), *dlerror*( ), *dlsym*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dlfcn.h>**

8579 **CHANGE HISTORY**

8580 First released in Issue 5.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

dlsym — obtain the address of a symbol from a *dlopen*( ) object

SYNOPSIS

XSI
```
#include <dlfcn.h>

void *dlsym(void *restrict handle, const char *restrict name);
```

DESCRIPTION

The *dlsym*( ) function shall obtain the address of a symbol defined within an object made accessible through a *dlopen*( ) call. The *handle* argument is the value returned from a call to *dlopen*( ) (and which has not since been released via a call to *dlclose*( )), and *name* is the symbol's name as a character string.

The *dlsym*( ) function shall search for the named symbol in all objects loaded automatically as a result of loading the object referenced by *handle* (see *dlopen*( )). Load ordering is used in *dlsym*( ) operations upon the global symbol object. The symbol resolution algorithm used shall be dependency order as described in *dlopen*( ).

The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

RETURN VALUE

If *handle* does not refer to a valid object opened by *dlopen*( ), or if the named symbol cannot be found within any of the objects associated with *handle*, *dlsym*( ) shall return NULL. More detailed diagnostic information shall be available through *dlerror*( ).

ERRORS

No errors are defined.

EXAMPLES

The following example shows how *dlopen*( ) and *dlsym*( ) can be used to access either function or data objects. For simplicity, error checking has been omitted.

```
void    *handle;
int     *iptr, (*fptr)(int);

/* open the needed object */
handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

/* find the address of function and data objects */
*(void **)(&fptr) = dlsym(handle, "my_function");
iptr = (int *)dlsym(handle, "my_object");

/* invoke function, passing value of integer as a parameter */
(*fptr)(*iptr);
```

APPLICATION USAGE

Special purpose values for *handle* are reserved for future use. These values and their meanings are:

RTLD_DEFAULT The symbol lookup happens in the normal global scope; that is, a search for a symbol using this handle would find the same definition as a direct use of this symbol in the program code.

RTLD_NEXT Specifies the next object after this one that defines *name*. *This one* refers to the object containing the invocation of *dlsym*( ). The *next* object is the one found upon the application of a load order symbol resolution algorithm (see *dlopen*( )). The next object is either one of global scope (because it was introduced as part of the original process image or because it was added with

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

a *dlopen*( ) operation including the RTLD_GLOBAL flag), or is an object that was included in the same *dlopen*( ) operation that loaded this one.

The RTLD_NEXT flag is useful to navigate an intentionally created hierarchy of multiply-defined symbols created through *interposition*. For example, if a program wished to create an implementation of *malloc*( ) that embedded some statistics gathering about memory allocations, such an implementation could use the real *malloc*( ) definition to perform the memory allocation—and itself only embed the necessary logic to implement the statistics gathering function.

RATIONALE

The ISO C standard does not require that pointers to functions can be cast back and forth to pointers to data. Indeed, the ISO C standard does not require that an object of type **void *** can hold a pointer to a function. Implementations supporting the XSI extension, however, do require that an object of type **void *** can hold a pointer to a function. The result of converting a pointer to a function into a pointer to another data type (except **void ***) is still undefined, however. Note that compilers conforming to the ISO C standard are required to generate a warning if a conversion from a **void *** pointer to a function pointer is attempted as in:

```
fptr = (int (*)(int))dlsym(handle, "my_function");
```

Due to the problem noted here, a future version may either add a new function to return function pointers, or the current interface may be deprecated in favor of two new functions: one that returns data pointers and the other that returns function pointers.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose( ), *dlerror*( ), *dlopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dlfcn.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The **restrict** keyword is added to the *dlsym*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 14 is applied, correcting an example, and adding text to the RATIONALE describing issues related to conversion of pointers to functions and back again.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS

XSI

```
#include <stdlib.h>

double drand48(void);
double erand48(unsigned short xsubi[3]);
long jrand48(unsigned short xsubi[3]);
void lcong48(unsigned short param[7]);
long lrand48(void);
long mrand48(void);
long nrand48(unsigned short xsubi[3]);
unsigned short *seed48(unsigned short seed16v[3]);
void srand48(long seedval);
```

DESCRIPTION

This family of functions shall generate pseudo-random numbers using a linear congruential algorithm and 48-bit integer arithmetic.

The *drand48*( ) and *erand48*( ) functions shall return non-negative, double-precision, floating-point values, uniformly distributed over the interval [0.0,1.0).

The *lrand48*( ) and *nrand48*( ) functions shall return non-negative, long integers, uniformly distributed over the interval $[0,2^{31})$.

The *mrand48*( ) and *jrand48*( ) functions shall return signed long integers uniformly distributed over the interval $[-2^{31},2^{31})$.

The *srand48*( ), *seed48*( ), and *lcong48*( ) functions are initialization entry points, one of which should be invoked before either *drand48*( ), *lrand48*( ), or *mrand48*( ) is called. (Although it is not recommended practice, constant default initializer values shall be supplied automatically if *drand48*( ), *lrand48*( ), or *mrand48*( ) is called without a prior call to an initialization entry point.) The *erand48*( ), *nrand48*( ), and *jrand48*( ) functions do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula:

$$X_{n+1} = (aX_n + c)_{\mathrm{mod}\ m} \qquad n \geq 0$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48*( ) is invoked, the multiplier value *a* and the addend value *c* are given by:

$$a = 5\mathrm{DEECE66D}_{16} = 273673163155_{8}$$

$$c = \mathrm{B}_{16} = 13_{8}$$

The value returned by any of the *drand48*( ), *erand48*( ), *jrand48*( ), *lrand48*( ), *mrand48*( ), or *nrand48*( ) functions is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of X_i and transformed into the returned value.

The *drand48*( ), *lrand48*( ), and *mrand48*( ) functions store the last 48-bit X_i generated in an internal buffer; that is why the application shall ensure that these are initialized prior to being invoked. The *erand48*( ), *nrand48*( ), and *jrand48*( ) functions require the calling program to provide storage for the successive X_i values in the array specified as an argument when the

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, *erand48*(), *nrand48*(), and *jrand48*() allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers; that is, the sequence of numbers in each stream shall *not* depend upon how many times the routines are called to generate numbers for the other streams.

The initializer function *srand48*() sets the high-order 32 bits of X_i to the low-order 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function *seed48*() sets the value of X_i to the 48-bit value specified in the argument array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[*0*]. The mid-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[*1*]. The high-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[*2*]. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by *seed48*(), and a pointer to this buffer is the value returned by *seed48*(). This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time—use the pointer to get at and store the last X_i value, and then use this value to reinitialize via *seed48*() when the program is restarted.

The initializer function *lcong48*() allows the user to specify the initial X_i, the multiplier value *a*, and the addend value *c*. Argument array elements *param*[*0-2*] specify X_i, *param*[*3-5*] specify the multiplier *a*, and *param*[*6*] specifies the 16-bit addend *c*. After *lcong48*() is called, a subsequent call to either *srand48*() or *seed48*() shall restore the standard multiplier and addend values, *a* and *c*, specified above.

The *drand48*(), *lrand48*(), and *mrand48*() functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

As described in the DESCRIPTION above.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

rand(), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8747 **NAME**

8748 dup, dup2 — duplicate an open file descriptor

8749 **SYNOPSIS**

8750
```
#include <unistd.h>
```

8751
```
int dup(int fildes);
```
8752
```
int dup2(int fildes, int fildes2);
```

8753 **DESCRIPTION**

8754 The *dup*( ) and *dup2*( ) functions provide an alternative interface to the service provided by
8755 *fcntl*( ) using the F_DUPFD command. The call:

8756
```
fid = dup(fildes);
```

8757 shall be equivalent to:

8758
```
fid = fcntl(fildes, F_DUPFD, 0);
```

8759 The call:

8760
```
fid = dup2(fildes, fildes2);
```

8761 shall be equivalent to:

8762
```
close(fildes2);
```
8763
```
fid = fcntl(fildes, F_DUPFD, fildes2);
```

8764 except for the following:

8765 8766
- If *fildes2* is less than 0 or greater than or equal to {OPEN_MAX}, *dup2*( ) shall return –1 with *errno* set to [EBADF].

8767 8768
- If *fildes* is a valid file descriptor and is equal to *fildes2*, *dup2*( ) shall return *fildes2* without closing it.

8769
- If *fildes* is not a valid file descriptor, *dup2*( ) shall return –1 and shall not close *fildes2*.

8770 8771
- The value returned shall be equal to the value of *fildes2* upon successful completion, or –1 upon failure.

8772 **RETURN VALUE**

8773 8774 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned; otherwise, –1 shall be returned and *errno* set to indicate the error.

8775 **ERRORS**

8776 The *dup*( ) function shall fail if:

8777 [EBADF] The *fildes* argument is not a valid open file descriptor.

8778 8779 [EMFILE] The number of file descriptors in use by this process would exceed {OPEN_MAX}.

8780 The *dup2*( ) function shall fail if:

8781 8782 [EBADF] The *fildes* argument is not a valid open file descriptor or the argument *fildes2* is negative or greater than or equal to {OPEN_MAX}.

8783 [EINTR] The *dup2*( ) function was interrupted by a signal.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Redirecting Standard Output to a File

The following example closes standard output for the current processes, re-assigns standard output to go to the file referenced by *pfd*, and closes the original file descriptor to clean up.

```
#include <unistd.h>
...
int pfd;
...
close(1);
dup(pfd);
close(pfd);
...
```

Redirecting Error Messages

The following example redirects messages from *stderr* to *stdout*.

```
#include <unistd.h>
...
dup2(1, 2);
...
```

APPLICATION USAGE

None.

RATIONALE

The *dup*( ) and *dup2*( ) functions are redundant. Their services are also provided by the *fcntl*( ) function. They have been included in this volume of IEEE Std 1003.1-2001 primarily for historical reasons, since many existing applications use them.

While the brief code segment shown is very similar in behavior to *dup2*( ), a conforming implementation based on other functions defined in this volume of IEEE Std 1003.1-2001 is significantly more complex. Least obvious is the possible effect of a signal-catching function that could be invoked between steps and allocate or deallocate file descriptors. This could be avoided by blocking signals.

The *dup2*( ) function is not marked obsolescent because it presents a type-safe version of functionality provided in a type-unsafe version by *fcntl*( ). It is used in the POSIX Ada binding.

The *dup2*( ) function is not intended for use in critical regions as a synchronization mechanism.

In the description of [EBADF], the case of *fildes* being out of range is covered by the given case of *fildes* not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter whether *fildes2* refers to an open file when the *dup2*( ) call is made.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *fcntl*( ), *open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8824 **CHANGE HISTORY**

8825 First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

ecvt, fcvt, gcvt — convert a floating-point number to a string (**LEGACY**)

SYNOPSIS

XSI
```
#include <stdlib.h>

char *ecvt(double value, int ndigit, int *restrict decpt,
    int *restrict sign);
char *fcvt(double value, int ndigit, int *restrict decpt,
    int *restrict sign);
char *gcvt(double value, int ndigit, char *buf);
```

DESCRIPTION

The *ecvt*( ), *fcvt*( ), and *gcvt*( ) functions shall convert floating-point numbers to null-terminated strings.

The *ecvt*( ) function shall convert *value* to a null-terminated string of *ndigit* digits (where *ndigit* is reduced to an unspecified limit determined by the precision of a **double**) and return a pointer to the string. The high-order digit shall be non-zero, unless the value is 0. The low-order digit shall be rounded in an implementation-defined manner. The position of the radix character relative to the beginning of the string shall be stored in the integer pointed to by *decpt* (negative means to the left of the returned digits). If *value* is zero, it is unspecified whether the integer pointed to by *decpt* would be 0 or 1. The radix character shall not be included in the returned string. If the sign of the result is negative, the integer pointed to by *sign* shall be non-zero; otherwise, it shall be 0.

If the converted value is out of range or is not representable, the contents of the returned string are unspecified.

The *fcvt*( ) function shall be equivalent to *ecvt*( ), except that *ndigit* specifies the number of digits desired after the radix character. The total number of digits in the result string is restricted to an unspecified limit as determined by the precision of a **double**.

The *gcvt*( ) function shall convert *value* to a null-terminated string (similar to that of the `%g` conversion specification format of *printf*( )) in the array pointed to by *buf* and shall return *buf*. It shall produce *ndigit* significant digits (limited to an unspecified value determined by the precision of a **double**) in the `%f` conversion specification format of *printf*( ) if possible, or the `%e` conversion specification format of *printf*( ) (scientific notation) otherwise. A minus sign shall be included in the returned string if *value* is less than 0. A radix character shall be included in the returned string if *value* is not a whole number. Trailing zeros shall be suppressed where *value* is not a whole number. The radix character is determined by the current locale. If *setlocale*( ) has not been called successfully, the default locale, POSIX, is used. The default locale specifies a period (`'.'`) as the radix character. The *LC_NUMERIC* category determines the value of the radix character within the current locale.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *ecvt*( ) and *fcvt*( ) functions shall return a pointer to a null-terminated string of digits.

The *gcvt*( ) function shall return *buf*.

The return values from *ecvt*( ) and *fcvt*( ) may point to static data which may be overwritten by subsequent calls to these functions.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *sprintf*( ) function is preferred over this function.

RATIONALE

None.

FUTURE DIRECTIONS

These functions may be withdrawn in a future version.

SEE ALSO

printf( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

Issue 6

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

This function is marked LEGACY.

The **restrict** keyword is added to the *ecvt*( ) and *fcvt*( ) prototypes for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION is updated to explicitly use ‘‘conversion specification’’ to describe `%g`, `%f`, and `%e`.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8896 **NAME**

8897 encrypt — encoding function (**CRYPT**)

8898 **SYNOPSIS**

8899 XSI
```
#include <unistd.h>
```

8900
```
void encrypt(char block[64], int edflag);
```

8901

8902 **DESCRIPTION**

8903 The *encrypt*( ) function shall provide access to an implementation-defined encoding algorithm.
8904 The key generated by *setkey*( ) is used to encrypt the string *block* with *encrypt*( ).

8905 The *block* argument to *encrypt*( ) shall be an array of length 64 bytes containing only the bytes
8906 with values of 0 and 1. The array is modified in place to a similar array using the key set by
8907 *setkey*( ). If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see
8908 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to
8909 [ENOSYS].

8910 The *encrypt*( ) function shall not change the setting of *errno* if successful. An application wishing
8911 to check for error situations should set *errno* to 0 before calling *encrypt*( ). If *errno* is non-zero on
8912 return, an error has occurred.

8913 The *encrypt*( ) function need not be reentrant. A function that is not required to be reentrant is
8914 not required to be thread-safe.

8915 **RETURN VALUE**

8916 The *encrypt*( ) function shall not return a value.

8917 **ERRORS**

8918 The *encrypt*( ) function shall fail if:

8919 [ENOSYS] The functionality is not supported on this implementation.

8920 **EXAMPLES**

8921 None.

8922 **APPLICATION USAGE**

8923 Historical implementations of the *encrypt*( ) function used a rather primitive encoding algorithm.

8924 In some environments, decoding might not be implemented. This is related to some Government
8925 restrictions on encryption and decryption routines. Historical practice has been to ship a
8926 different version of the encryption library without the decryption feature in the routines
8927 supplied. Thus the exported version of *encrypt*( ) does encoding but not decoding.

8928 **RATIONALE**

8929 None.

8930 **FUTURE DIRECTIONS**

8931 None.

8932 **SEE ALSO**

8933 *crypt*( ), *setkey*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

8934 **CHANGE HISTORY**

8935 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8936 **Issue 5**

8937 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8938 **Issue 6**

8939 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

endgrent, getgrent, setgrent — group database entry functions

SYNOPSIS

XSI
```
#include <grp.h>

void endgrent(void);
struct group *getgrent(void);
void setgrent(void);
```

DESCRIPTION

The *getgrent*( ) function shall return a pointer to a structure containing the broken-out fields of an entry in the group database. When first called, *getgrent*( ) shall return a pointer to a **group** structure containing the first entry in the group database. Thereafter, it shall return a pointer to a **group** structure containing the next group structure in the group database, so successive calls may be used to search the entire database.

An implementation that provides extended security controls may impose further implementation-defined restrictions on accessing the group database. In particular, the system may deny the existence of some or all of the group database entries associated with groups other than those groups associated with the caller and may omit users other than the caller from the list of members of groups in database entries that are returned.

The *setgrent*( ) function shall rewind the group database to allow repeated searches.

The *endgrent*( ) function may be called to close the group database when processing is complete.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

When first called, *getgrent*( ) shall return a pointer to the first group structure in the group database. Upon subsequent calls it shall return the next group structure in the group database. The *getgrent*( ) function shall return a null pointer on end-of-file or an error and *errno* may be set to indicate the error.

The return value may point to a static area which is overwritten by a subsequent call to *getgrgid*( ), *getgrnam*( ), or *getgrent*( ).

ERRORS

The *getgrent*( ) function may fail if:

[EINTR]	A signal was caught during the operation.
[EIO]	An I/O error has occurred.
[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
[ENFILE]	The maximum allowable number of files is currently open in the system.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8976 **EXAMPLES**

8977 None.

8978 **APPLICATION USAGE**

8979 These functions are provided due to their historical usage. Applications should avoid
8980 dependencies on fields in the group database, whether the database is a single file, or where in
8981 the file system name space the database resides. Applications should use *getgrnam*( ) and
8982 *getgrgid*( ) whenever possible because it avoids these dependencies.

8983 **RATIONALE**

8984 None.

8985 **FUTURE DIRECTIONS**

8986 None.

8987 **SEE ALSO**

8988 *getgrgid*( ), *getgrnam*( ), *getlogin*( ), *getpwent*( ), the Base Definitions volume of
8989 IEEE Std 1003.1-2001, **<grp.h>**

8990 **CHANGE HISTORY**

8991 First released in Issue 4, Version 2.

8992 **Issue 5**

8993 Moved from X/OPEN UNIX extension to BASE.

8994 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
8995 VALUE section.

8996 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8997 **Issue 6**

8998 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8999 **NAME**

9000 endhostent, gethostent, sethostent — network host database functions

9001 **SYNOPSIS**

9002
```
#include <netdb.h>
```

9003
```
void endhostent(void);
```
9004
```
struct hostent *gethostent(void);
```
9005
```
void sethostent(int stayopen);
```

9006 **DESCRIPTION**

9007 These functions shall retrieve information about hosts. This information is considered to be
9008 stored in a database that can be accessed sequentially or randomly. The implementation of this
9009 database is unspecified.

9010 **Note:** In many cases this database is implemented by the Domain Name System, as documented in
9011 RFC 1034, RFC 1035, and RFC 1886.

9012 The *sethostent*( ) function shall open a connection to the database and set the next entry for
9013 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection
9014 shall not be closed by a call to *gethostent*( ), *gethostbyname*( ), or *gethostbyaddr*( ), and the
9015 implementation may maintain an open file descriptor.

9016 The *gethostent*( ) function shall read the next entry in the database, opening and closing a
9017 connection to the database as necessary.

9018 Entries shall be returned in **hostent** structures. Refer to *gethostbyaddr*( ) for a definition of the
9019 **hostent** structure.

9020 The *endhostent*( ) function shall close the connection to the database, releasing any open file
9021 descriptor.

9022 These functions need not be reentrant. A function that is not required to be reentrant is not
9023 required to be thread-safe.

9024 **RETURN VALUE**

9025 Upon successful completion, the *gethostent*( ) function shall return a pointer to a **hostent**
9026 structure if the requested entry was found, and a null pointer if the end of the database was
9027 reached or the requested entry was not found.

9028 **ERRORS**

9029 No errors are defined for *endhostent*( ), *gethostent*( ), and *sethostent*( ).

9030 **EXAMPLES**

9031 None.

9032 **APPLICATION USAGE**

9033 The *gethostent*( ) function may return pointers to static data, which may be overwritten by
9034 subsequent calls to any of these functions.

9035 **RATIONALE**

9036 None.

9037 **FUTURE DIRECTIONS**

9038 None.

9039 **SEE ALSO**

9040 *endservent*( ), *gethostbyaddr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<netdb.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9041 **CHANGE HISTORY**

9042 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

SYNOPSIS

```
#include <netdb.h>

void endnetent(void);
struct netent *getnetbyaddr(uint32_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
void setnetent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about networks. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setnetent*( ) function shall open and rewind the database. If the *stayopen* argument is non-zero, the connection to the *net* database shall not be closed after each call to *getnetent*( ) (either directly, or indirectly through one of the other *getnet**( ) functions), and the implementation may maintain an open file descriptor to the database.

The *getnetent*( ) function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getnetbyaddr*( ) function shall search the database from the beginning, and find the first entry for which the address family specified by *type* matches the *n_addrtype* member and the network number *net* matches the *n_net* member, opening and closing a connection to the database as necessary. The *net* argument shall be the network number in host byte order.

The *getnetbyname*( ) function shall search the database from the beginning and find the first entry for which the network name specified by *name* matches the *n_name* member, opening and closing a connection to the database as necessary.

The *getnetbyaddr*( ), *getnetbyname*( ), and *getnetent*( ) functions shall each return a pointer to a **netent** structure, the members of which shall contain the fields of an entry in the network database.

The *endnetent*( ) function shall close the database, releasing any open file descriptor.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *getnetbyaddr*( ), *getnetbyname*( ), and *getnetent*( ) shall return a pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The *getnetbyaddr*( ), *getnetbyname*( ), and *getnetent*( ) functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<netdb.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol database functions

SYNOPSIS

```
#include <netdb.h>

void endprotoent(void);
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
void setprotoent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about protocols. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setprotoent*( ) function shall open a connection to the database, and set the next entry to the first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database shall not be closed after each call to *getprotoent*( ) (either directly, or indirectly through one of the other *getproto**( ) functions), and the implementation may maintain an open file descriptor for the database.

The *getprotobyname*( ) function shall search the database from the beginning and find the first entry for which the protocol name specified by *name* matches the *p_name* member, opening and closing a connection to the database as necessary.

The *getprotobynumber*( ) function shall search the database from the beginning and find the first entry for which the protocol number specified by *proto* matches the *p_proto* member, opening and closing a connection to the database as necessary.

The *getprotoent*( ) function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getprotobyname*( ), *getprotobynumber*( ), and *getprotoent*( ) functions shall each return a pointer to a **protoent** structure, the members of which shall contain the fields of an entry in the network protocol database.

The *endprotoent*( ) function shall close the connection to the database, releasing any open file descriptor.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *getprotobyname*( ), *getprotobynumber*( ), and *getprotoent*( ) return a pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

ERRORS

No errors are defined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The *getprotobyname*(), *getprotobynumber*(), and *getprotoent*() functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<netdb.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

endpwent, getpwent, setpwent — user database functions

SYNOPSIS

XSI
```
#include <pwd.h>

void endpwent(void);
struct passwd *getpwent(void);
void setpwent(void);
```

DESCRIPTION

These functions shall retrieve information about users.

The *getpwent*( ) function shall return a pointer to a structure containing the broken-out fields of an entry in the user database. Each entry in the user database contains a **passwd** structure. When first called, *getpwent*( ) shall return a pointer to a **passwd** structure containing the first entry in the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next entry in the user database. Successive calls can be used to search the entire user database.

If an end-of-file or an error is encountered on reading, *getpwent*( ) shall return a null pointer.

An implementation that provides extended security controls may impose further implementation-defined restrictions on accessing the user database. In particular, the system may deny the existence of some or all of the user database entries associated with users other than the caller.

The *setpwent*( ) function effectively rewinds the user database to allow repeated searches.

The *endpwent*( ) function may be called to close the user database when processing is complete.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *getpwent*( ) function shall return a null pointer on end-of-file or error.

ERRORS

The *getpwent*( ), *setpwent*( ), and *endpwent*( ) functions may fail if:

[EIO] An I/O error has occurred.

In addition, *getpwent*( ) and *setpwent*( ) may fail if:

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENFILE] The maximum allowable number of files is currently open in the system.

The return value may point to a static area which is overwritten by a subsequent call to *getpwuid*( ), *getpwnam*( ), or *getpwent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Searching the User Database

The following example uses the *getpwent*() function to get successive entries in the user database, returning a pointer to a **passwd** structure that contains information about each user. The call to *endpwent*() closes the user database and cleans up.

```
#include <pwd.h>
...
struct passwd *p;
...
while ((p = getpwent ()) != NULL) {
...
}

endpwent();
...
```

APPLICATION USAGE

These functions are provided due to their historical usage. Applications should avoid dependencies on fields in the password database, whether the database is a single file, or where in the file system name space the database resides. Applications should use *getpwuid*() whenever possible because it avoids these dependencies.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent(), *getlogin*(), *getpwnam*(), *getpwuid*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<pwd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

Issue 6

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9218 **NAME**

9219 endservent, getservbyname, getservbyport, getservent, setservent — network services database
9220 functions

9221 **SYNOPSIS**

9222 ```
#include <netdb.h>
```

9223 ```
void endservent(void);
```
9224 ```
struct servent *getservbyname(const char *name, const char *proto);
```
9225 ```
struct servent *getservbyport(int port, const char *proto);
```
9226 ```
struct servent *getservent(void);
```
9227 ```
void setservent(int stayopen);
```

9228 **DESCRIPTION**

9229 These functions shall retrieve information about network services. This information is
9230 considered to be stored in a database that can be accessed sequentially or randomly. The
9231 implementation of this database is unspecified.

9232 The *setservent*( ) function shall open a connection to the database, and set the next entry to the
9233 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each
9234 call to the *getservent*( ) function (either directly, or indirectly through one of the other *getserv**( )
9235 functions), and the implementation may maintain an open file descriptor for the database.

9236 The *getservent*( ) function shall read the next entry of the database, opening and closing a
9237 connection to the database as necessary.

9238 The *getservbyname*( ) function shall search the database from the beginning and find the first
9239 entry for which the service name specified by *name* matches the *s_name* member and the protocol
9240 name specified by *proto* matches the *s_proto* member, opening and closing a connection to the
9241 database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be
9242 matched.

9243 The *getservbyport*( ) function shall search the database from the beginning and find the first entry
9244 for which the port specified by *port* matches the *s_port* member and the protocol name specified
9245 by *proto* matches the *s_proto* member, opening and closing a connection to the database as
9246 necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port*
9247 argument shall be in network byte order.

9248 The *getservbyname*( ), *getservbyport*( ), and *getservent*( ) functions shall each return a pointer to a
9249 **servent** structure, the members of which shall contain the fields of an entry in the network
9250 services database.

9251 The *endservent*( ) function shall close the database, releasing any open file descriptor.

9252 These functions need not be reentrant. A function that is not required to be reentrant is not
9253 required to be thread-safe.

9254 **RETURN VALUE**

9255 Upon successful completion, *getservbyname*( ), *getservbyport*( ), and *getservent*( ) return a pointer to
9256 a **servent** structure if the requested entry was found, and a null pointer if the end of the database
9257 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

9258 **ERRORS**

9259 No errors are defined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9260 **EXAMPLES**

9261 None.

9262 **APPLICATION USAGE**

9263 The *port* argument of *getservbyport*( ) need not be compatible with the port values of all address
9264 families.

9265 The *getservbyname*( ), *getservbyport*( ), and *getservent*( ) functions may return pointers to static
9266 data, which may be overwritten by subsequent calls to any of these functions.

9267 **RATIONALE**

9268 None.

9269 **FUTURE DIRECTIONS**

9270 None.

9271 **SEE ALSO**

9272 *endhostent*( ), *endprotoent*( ), *htonl*( ), *inet_addr*( ), the Base Definitions volume of
9273 IEEE Std 1003.1-2001, **<netdb.h>**

9274 **CHANGE HISTORY**

9275 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database functions

SYNOPSIS

XSI
```
#include <utmpx.h>

void endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
struct utmpx *pututxline(const struct utmpx *utmpx);
void setutxent(void);
```

DESCRIPTION

These functions shall provide access to the user accounting database.

The *getutxent*( ) function shall read the next entry from the user accounting database. If the database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

The *getutxid*( ) function shall search forward from the current point in the database. If the *ut_type* value of the **utmpx** structure pointed to by *id* is BOOT_TIME, OLD_TIME, or NEW_TIME, then it shall stop when it finds an entry with a matching *ut_type* value. If the *ut_type* value is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is reached without a match, *getutxid*( ) shall fail.

The *getutxline*( ) function shall search forward from the current point in the database until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a *ut_line* value matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached without a match, *getutxline*( ) shall fail.

The *getutxid*( ) or *getutxline*( ) function may cache data. For this reason, to use *getutxline*( ) to search for multiple occurrences, the application shall zero out the static data after each success, or *getutxline*( ) may return a pointer to the same **utmpx** structure.

There is one exception to the rule about clearing the structure before further reads are done. The implicit read done by *pututxline*( ) (if it finds that it is not already at the correct place in the user accounting database) shall not modify the static structure returned by *getutxent*( ), *getutxid*( ), or *getutxline*( ), if the application has modified this structure and passed the pointer back to *pututxline*( ).

For all entries that match a request, the *ut_type* member indicates the type of the entry. Other members of the entry shall contain meaningful data based on the value of the *ut_type* member as follows:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	*ut_tv*
OLD_TIME	*ut_tv*
NEW_TIME	*ut_tv*
USER_PROCESS	*ut_id*, *ut_user* (login name of the user), *ut_line*, *ut_pid*, *ut_tv*
INIT_PROCESS	*ut_id*, *ut_pid*, *ut_tv*
LOGIN_PROCESS	*ut_id*, *ut_user* (implementation-defined name of the login process), *ut_pid*, *ut_tv*
DEAD_PROCESS	*ut_id*, *ut_pid*, *ut_tv*

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

If the process has appropriate privileges, the *pututxline*( ) function shall write out the structure into the user accounting database. It shall use *getutxid*( ) to search for a record that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

The *endutxent*( ) function shall close the user accounting database.

The *setutxent*( ) function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *getutxent*( ), *getutxid*( ), and *getutxline*( ) shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

The return value may point to a static area which is overwritten by a subsequent call to *getutxid*( ) or *getutxline*( ).

Upon successful completion, *pututxline*( ) shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

The *endutxent*( ) and *setutxent*( ) functions shall not return a value.

ERRORS

No errors are defined for the *endutxent*( ), *getutxent*( ), *getutxid*( ), *getutxline*( ), and *setutxent*( ) functions.

The *pututxline*( ) function may fail if:

[EPERM] The process does not have appropriate privileges.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9353 **EXAMPLES**

9354 None.

9355 **APPLICATION USAGE**

9356 The sizes of the arrays in the structure can be found using the *sizeof* operator.

9357 **RATIONALE**

9358 None.

9359 **FUTURE DIRECTIONS**

9360 None.

9361 **SEE ALSO**

9362 The Base Definitions volume of IEEE Std 1003.1-2001, **<utmpx.h>**

9363 **CHANGE HISTORY**

9364 First released in Issue 4, Version 2.

9365 **Issue 5**

9366 Moved from X/OPEN UNIX extension to BASE.

9367 Normative text previously in the APPLICATION USAGE section is moved to the
9368 DESCRIPTION.

9369 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9370 **Issue 6**

9371 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

environ — array of character pointers to the environment strings

SYNOPSIS

```
extern char **environ;
```

DESCRIPTION

Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables and *exec*.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9379 **NAME**

9380 erand48 — generate uniformly distributed pseudo-random numbers

9381 **SYNOPSIS**

9382 XSI
```
#include <stdlib.h>
```

9383
```
double erand48(unsigned short xsubi[3]);
```

9384

9385 **DESCRIPTION**

9386 Refer to *drand48*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

erf, erff, erfl — error functions

SYNOPSIS

```
#include <math.h>

double erf(double x);
float erff(float x);
long double erfl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the error function of their argument *x*, defined as:

$$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt$$

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of the error function.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, ±0 shall be returned.

If *x* is ±Inf, ±1 shall be returned.

If *x* is subnormal, a range error may occur, and 2 * *x* /*sqrt*(π) should be returned.

ERRORS

These functions may fail if:

MX

Range Error	The result underflows.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

Underflow occurs when $|x| < \text{DBL_MIN} * (sqrt(\pi)/2)$.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

erfc( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *erf*( ) function is no longer marked as an extension.

The *erfc*( ) function is split out onto its own reference page.

The *erff*( ) and *erfl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

erfc, erfcf, erfcl — complementary error functions

SYNOPSIS

```
#include <math.h>

double erfc(double x);
float erfcf(float x);
long double erfcl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the complementary error function 1.0 – *erf*(*x*).

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of the complementary error function.

If the correct value would cause underflow and is not representable, a range error may occur and MX either 0.0 (if representable), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, +1 shall be returned.

If *x* is –Inf, +2 shall be returned.

If *x* is +Inf, +0 shall be returned.

If the correct value would cause underflow and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions may fail if:

Range Error — The result underflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

The *erfc*( ) function is provided because of the extreme loss of relative accuracy if *erf*(*x*) is called for large *x* and the result subtracted from 1.0.

Note for IEEE Std 754-1985 **double**, 26.55 < *x* implies *erfc*(*x*) has underflowed.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

erf( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *erfc*( ) function is no longer marked as an extension.

These functions are split out from the *erf*( ) reference page.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

9504 **NAME**

9505 erff, erfl — error functions

9506 **SYNOPSIS**

9507 `#include <math.h>`

9508 `float erff(float x);`

9509 `long double erfl(long double x);`

9510 **DESCRIPTION**

9511 Refer to *erf*( ).

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

errno — error return value

SYNOPSIS

```
#include <errno.h>
```

DESCRIPTION

The lvalue *errno* is used by many functions to return error values.

Many functions provide an error number in *errno*, which has type **int** and is defined in **<errno.h>**. The value of *errno* shall be defined only after a call to a function for which it is explicitly stated to be set and until it is changed by the next function call or if the application assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by a function's return value. Applications shall obtain the definition of *errno* by the inclusion of **<errno.h>**. No function in this volume of IEEE Std 1003.1-2001 shall set *errno* to 0.

It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant pages.

RETURN VALUE

None.

ERRORS

None.

EXAMPLES

None.

APPLICATION USAGE

Previously both POSIX and X/Open documents were more restrictive than the ISO C standard in that they required *errno* to be defined as an external variable, whereas the ISO C standard required only that *errno* be defined as a modifiable lvalue with type **int**.

An application that needs to examine the value of *errno* to determine the error should set it to 0 before a function call, then inspect it before a subsequent function call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.3, the Base Definitions volume of IEEE Std 1003.1-2001, **<errno.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The following sentence is deleted from the DESCRIPTION: ''The value of *errno* is 0 at program start-up, but is never set to 0 by any XSI function''. The DESCRIPTION also no longer states that conforming implementations may support the declaration:

```
extern int errno;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

Obsolescent text regarding defining *errno* as:

```
extern int errno
```

is removed.

Text regarding no function setting *errno* to zero to indicate an error is changed to no function shall set *errno* to zero. This is for alignment with the ISO / IEC 9899: 1999 standard.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

environ, execl, execv, execle, execve, execlp, execvp — execute a file

SYNOPSIS

```
#include <unistd.h>

extern char **environ;
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execle(const char *path, const char *arg0, ... /*,
    (char *)0, char *const envp[]*/);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execvp(const char *file, char *const argv[]);
```

DESCRIPTION

The *exec* family of functions shall replace the current process image with a new process image. The new image shall be constructed from a regular, executable file called the *new process image file*. There shall be no return from a successful *exec*, because the calling process image is overlaid by the new process image.

When a C-language program is executed as a result of this call, it shall be entered as a C-language function call as follows:

```
int main (int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
extern char **environ;
```

is initialized as a pointer to an array of character pointers to the environment strings. The *argv* and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv* array is not counted in *argc*.

THR Conforming multi-threaded applications shall not use the *environ* variable to access or modify any environment variable while any other thread is concurrently modifying any environment variable. A call to any function dependent on any environment variable shall be considered a use of the *environ* variable to access that environment variable.

The arguments specified by a program with one of the *exec* functions shall be passed on to the new process image in the corresponding *main*( ) arguments.

The argument *path* points to a pathname that identifies the new process image file.

The argument *file* is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, the *file* argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable *PATH* (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If this environment variable is not present, the results of the search are implementation-defined.

There are two distinct ways in which the contents of the process image file may cause the execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the ERRORS section). In the cases where the other members of the *exec* family of functions would fail and set *errno* to [ENOEXEC], the *execlp*( ) and *execvp*( ) functions shall execute a command interpreter and the environment of the executed command shall be as if the process invoked the *sh* utility using *execl*( ) as follows:

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
execl(<shell path>, arg0, file, arg1, ..., (char *)0);
```

where <*shell path*> is an unspecified pathname for the *sh* utility, *file* is the process image file, and for *execvp*( ), where *arg*0, *arg*1, and so on correspond to the values passed to *execvp*( ) in *argv*[0], *argv*[1], and so on.

The arguments represented by *arg0*,... are pointers to null-terminated character strings. These strings shall constitute the argument list available to the new process image. The list is terminated by a null pointer. The argument *arg0* should point to a filename that is associated with the process being started by one of the *exec* functions.

The argument *argv* is an array of character pointers to null-terminated strings. The application shall ensure that the last member of this array is a null pointer. These strings shall constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process being started by one of the *exec* functions.

The argument *envp* is an array of character pointers to null-terminated strings. These strings shall constitute the environment for the new process image. The *envp* array is terminated by a null pointer.

For those forms not containing an *envp* pointer (*execl*( ), *execv*( ), *execlp*( ), and *execvp*( )), the environment for the new process image shall be taken from the external variable *environ* in the calling process.

The number of bytes available for the new process' combined argument and environment lists is {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any alignment bytes are included in this total.

File descriptors open in the calling process image shall remain open in the new process image, except for those whose close-on-*exec* flag FD_CLOEXEC is set. For those file descriptors that remain open, all attributes of the open file description remain unchanged. For any file descriptor that is closed for this reason, file locks are removed as a result of the close as described in *close*( ). Locks that are not removed by closing of file descriptors remain unchanged.

If file descriptors 0, 1, and 2 would otherwise be closed after a successful call to one of the *exec* family of functions, and the new process image file has the set-user-ID or set-group-ID file mode bits set, and the ST_NOSUID bit is not set for the file system containing the new process image file, implementations may open an unspecified file for each of these file descriptors in the new process image.

Directory streams open in the calling process image shall be closed in the new process image.

The state of the floating-point environment in the new process image shall be set to the default.

The state of conversion descriptors and message catalog descriptors in the new process image is undefined. For the new process image, the equivalent of:

```
setlocale(LC_ALL, "C")
```

shall be executed at start-up.

Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by the calling process image shall be set to be ignored by the new process image. Signals set to be caught by the calling process image shall be set to the default action in the new process image (see **<signal.h>**). If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the new process image. After a successful call to any of the *exec* functions, alternate signal stacks are not preserved and the SA_ONSTACK flag shall be cleared for all signals.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

After a successful call to any of the *exec* functions, any functions previously registered by *atexit*( ) are no longer registered.

XSI If the ST_NOSUID bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the new process image shall be set to the user ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image shall be set to the group ID of the new process image file. The real user ID, real group ID, and supplementary group IDs of the new process image shall remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image shall be saved (as the saved set-user-ID and the saved set-group-ID) for use by *setuid*( ).

XSI Any shared memory segments attached to the calling process image shall not be attached to the new process image.

SEM Any named semaphores open in the calling process shall be closed as if by appropriate calls to *sem_close*( ).

TYM Any blocks of typed memory that were mapped in the calling process are unmapped, as if *munmap*( ) was implicitly called to unmap them.

ML Memory locks established by the calling process via calls to *mlockall*( ) or *mlock*( ) shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to the *exec* function. If the *exec* function fails, the effect on memory locks is unspecified.

MF|SHM Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image.

PS For the SCHED_FIFO and SCHED_RR scheduling policies, the policy and priority settings shall not be changed by a call to an *exec* function. For other scheduling policies, the policy and priority settings on *exec* are implementation-defined.

TMR Per-process timers created by the calling process shall be deleted before replacing the current process image with the new process image.

MSG All open message queue descriptors in the calling process shall be closed, as described in *mq_close*( ).

AIO Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the *exec* function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the *exec* function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the *exec* function be affected by the presence of outstanding asynchronous I/O operations at the time the *exec* function is called. Whether any I/O is canceled, and which I/O may be canceled upon *exec*, is implementation-defined.

CPT The new process image shall inherit the CPU-time clock of the calling process image. This inheritance means that the process CPU-time clock of the process being *exec*-ed shall not be reinitialized or altered as a result of the *exec* function other than to reflect the time spent by the process executing the *exec* function itself.

TCT The initial value of the CPU-time clock of the initial thread of the new process image shall be set to zero.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TRC If the calling process is being traced, the new process image shall continue to be traced into the same trace stream as the original process image, but the new process image shall not inherit the mapping of trace event names to trace event type identifiers that was defined by calls to the *posix_trace_eventid_open*( ) or the *posix_trace_trid_eventid_open*( ) functions in the calling process image.

If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described in the *posix_trace_shutdown*( ) function.

The new process shall inherit at least the following attributes from the calling process image:

XSI • Nice value (see *nice*( ))

XSI • *semadj* values (see *semop*( ))

• Process ID

• Parent process ID

• Process group ID

• Session membership

• Real user ID

• Real group ID

• Supplementary group IDs

• Time left until an alarm clock signal (see *alarm*( ))

• Current working directory

• Root directory

• File mode creation mask (see *umask*( ))

XSI • File size limit (see *ulimit*( ))

• Process signal mask (see *sigprocmask*( ))

• Pending signal (see *sigpending*( ))

• *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* (see *times*( ))

XSI • Resource limits

XSI • Controlling terminal

XSI • Interval timers

All other process attributes defined in this volume of IEEE Std 1003.1-2001 shall be the same in the new and old process images. The inheritance of process attributes not defined by this volume of IEEE Std 1003.1-2001 is implementation-defined.

A call to any *exec* function from a process with more than one thread shall result in all threads being terminated and the new executable image being loaded and executed. No destructor functions shall be called.

Upon successful completion, the *exec* functions shall mark for update the *st_atime* field of the file. If an *exec* function failed but was able to locate the process image file, whether the *st_atime* field is marked for update is unspecified. Should the *exec* function succeed, the process image file shall be considered to have been opened with *open*( ). The corresponding *close*( ) shall be considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the *exec* functions, *posix_spawn*( ), or *posix_spawnp*( ).

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *argv*[ ] and *envp*[ ] arrays of pointers and the strings to which those arrays point shall not be modified by a call to one of the *exec* functions, except as a consequence of replacing the process image.

XSI The saved resource limits in the new process image are set to be a copy of the process' corresponding hard and soft limits.

RETURN VALUE

If one of the *exec* functions returns to the calling process image, an error has occurred; the return value shall be −1, and *errno* shall be set to indicate the error.

ERRORS

The *exec* functions shall fail if:

[E2BIG] The number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of {ARG_MAX} bytes.

[EACCES] Search permission is denied for a directory listed in the new process image file's path prefix, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.

[EINVAL] The new process image file has the appropriate permission and has a recognized executable binary format, but the system does not support execution of a file with this format.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file* argument.

[ENAMETOOLONG] The length of the *path* or *file* arguments exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an empty string.

[ENOTDIR] A component of the new process image file's path prefix is not a directory.

The *exec* functions, except for *execlp*( ) and *execvp*( ), shall fail if:

[ENOEXEC] The new process image file has the appropriate access permission but has an unrecognized format.

The *exec* functions may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* or *file* argument.

[ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

[ENOMEM] The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.

[ETXTBSY] The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Using execl()

The following example executes the *ls* command, specifying the pathname of the executable (**/bin/ls**) and using arguments supplied directly to the command to produce single-column output.

```
#include <unistd.h>

int ret;
...
ret = execl ("/bin/ls", "ls", "-1", (char *)0);
```

Using execle()

The following example is similar to **Using execl()**. In addition, it specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>

int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

Using execlp()

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable.

```
#include <unistd.h>

int ret;
...
ret = execlp ("ls", "ls", "-l", (char *)0);
```

Using execv()

The following example passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execv ("/bin/ls", cmd);
```


--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Using execve()

The following example passes arguments to the *ls* command in the *cmd* array, and specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execve ("/bin/ls", cmd, env);
```

Using execvp()

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable, and passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execvp ("ls", cmd);
```

APPLICATION USAGE

As the state of conversion descriptors and message catalog descriptors in the new process image is undefined, conforming applications should not rely on their use and should close them prior to calling one of the *exec* functions.

Applications that require other than the default POSIX locale should call *setlocale*() with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

Applications should not depend on file descriptors 0, 1, and 2 being closed after an *exec*. A future version may allow these file descriptors to be automatically opened for any process.

RATIONALE

Early proposals required that the value of *argc* passed to *main*() be ''one or greater''. This was driven by the same requirement in drafts of the ISO C standard. In fact, historical implementations have passed a value of zero when no arguments are supplied to the caller of the *exec* functions. This requirement was removed from the ISO C standard and subsequently removed from this volume of IEEE Std 1003.1-2001 as well. The wording, in particular the use of the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an application. In fact, this is good practice, since many existing applications reference *argv*[0] without first checking the value of *argc*.

The requirement on a Strictly Conforming POSIX Application also states that the value passed as the first argument be a filename associated with the process being started. Although some existing applications pass a pathname rather than a filename in some circumstances, a filename is more generally useful, since the common usage of *argv*[0] is in printing diagnostics. In some cases the filename passed is not the actual filename of the file; for example, many implementations of the *login* utility use a convention of prefixing a hyphen ('−') to the actual filename, which indicates to the command interpreter being invoked that it is a ''login shell''.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Historically there have been two ways that implementations can *exec* shell scripts.

One common historical implementation is that the *execl*( ), *execv*( ), *execle*( ), and *execve*( ) functions return an [ENOEXEC] error for any file not recognizable as executable, including a shell script. When the *execlp*( ) and *execvp*( ) functions encounter such a file, they assume the file to be a shell script and invoke a known command interpreter to interpret such files. This is now required by IEEE Std 1003.1-2001. These implementations of *execvp*( ) and *execlp*( ) only give the [ENOEXEC] error in the rare case of a problem with the command interpreter's executable file. Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp*( ) or *execvp*( ), although implementations can still give it.

Another way that some historical implementations handle shell scripts is by recognizing the first two bytes of the file as the character string "#!" and using the remainder of the first line of the file as the name of the command interpreter to execute.

One potential source of confusion noted by the standard developers is over how the contents of a process image file affect the behavior of the *exec* family of functions. The following is a description of the actions taken:

1. If the process image file is a valid executable (in a format that is executable and valid and having appropriate permission) for this system, then the system executes the file.
2. If the process image file has appropriate permission and is in a format that is executable but not valid for this system (such as a recognized binary for another architecture), then this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
3. If the process image file has appropriate permission but is not otherwise recognized:
   a. If this is a call to *execlp*( ) or *execvp*( ), then they invoke a command interpreter assuming that the process image file is a shell script.
   b. If this is not a call to *execlp*( ) or *execvp*( ), then an error occurs and *errno* is set to [ENOEXEC].

Applications that do not require to access their arguments may use the form:

```
main(void)
```

as specified in the ISO C standard. However, the implementation will always provide the two arguments *argc* and *argv*, even if they are not used.

Some implementations provide a third argument to *main*( ) called *envp*. This is defined as a pointer to the environment. The ISO C standard specifies invoking *main*( ) with two arguments, so implementations must support applications written this way. Since this volume of IEEE Std 1003.1-2001 defines the global variable *environ*, which is also provided by historical implementations and can be used anywhere that *envp* could be used, there is no functional need for the *envp* argument. Applications should use the *getenv*( ) function rather than accessing the environment directly via either *envp* or *environ*. Implementations are required to support the two-argument calling sequence, but this does not prohibit an implementation from supporting *envp* as an optional third argument.

This volume of IEEE Std 1003.1-2001 specifies that signals set to SIG_IGN remain set to SIG_IGN, and that the process signal mask be unchanged across an *exec*. This is consistent with historical implementations, and it permits some useful functionality, such as the *nohup* command. However, it should be noted that many existing applications wrongly assume that they start with certain signals set to the default action and/or unblocked. In particular, applications written with a simpler signal model that does not include blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked with some signals blocked. Therefore, it is best not to block or ignore signals across *exec*s without explicit reason to do so,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

and especially not to block signals across *exec*s of arbitrary (not closely co-operating) programs.

The *exec* functions always save the value of the effective user ID and effective group ID of the process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of the process image file is set.

The statement about *argv*[ ] and *envp*[ ] being constants is included to make explicit to future writers of language bindings that these objects are completely constant. Due to a limitation of the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of *const–qualification* for the *argv*[ ] and *envp*[ ] parameters for the *exec* functions may seem to be the natural choice, given that these functions do not modify either the array of pointers or the characters to which the function points, but this would disallow existing correct code. Instead, only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src* derived from the ISO C standard summarizes the compatibility:

dst:	**char *[ ]**	**const char *[ ]**	**char *const[ ]**	**const char *const[ ]**
src:				
char *[ ]	VALID	—	VALID	—
const char *[ ]	—	VALID	—	VALID
char * const [ ]	—	—	VALID	—
const char *const[ ]	—	—	—	VALID

Since all existing code has a source type matching the first row, the column that gives the most valid combinations is the third column. The only other possibility is the fourth column, but using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth column cannot be used, because the declaration a non-expert would naturally use would be that in the second row.

The ISO C standard and this volume of IEEE Std 1003.1-2001 do not conflict on the use of *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in the same way as an entry point (for example, *fork*( )), it conforms to both standards. A library can contain *fork*( ), but if there is a user-provided *fork*( ), that *fork*( ) is given precedence and no problem ensues. The situation is similar for *environ*: the definition in this volume of IEEE Std 1003.1-2001 is to be used if there is no user-provided *environ* to take precedence. At least three implementations are known to exist that solve this problem.

[E2BIG] The limit {ARG_MAX} applies not just to the size of the argument list, but to the sum of that and the size of the environment list.

[EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the new process image file is corrupted. They are non-conforming.

[EINVAL] This error condition was added to IEEE Std 1003.1-2001 to allow an implementation to detect executable files generated for different architectures, and indicate this situation to the application. Historical implementations of shells, *execvp*( ), and *execlp*( ) that encounter an [ENOEXEC] error will execute a shell on the assumption that the file is a shell script. This will not produce the desired effect when the file is a valid executable for a different architecture. An implementation may now choose to avoid this problem by returning [EINVAL] when a valid executable for a different architecture is encountered. Some historical implementations return [EINVAL] to indicate that the *path* argument contains a character with the high order bit set. The standard developers chose to deviate from historical practice for the following reasons:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1. The new utilization of [EINVAL] will provide some measure of utility to the user community.
2. Historical use of [EINVAL] is not acceptable in an internationalized operating environment.

[ENAMETOOLONG]
Since the file pathname may be constructed by taking elements in the *PATH* variable and putting them together with the filename, the [ENAMETOOLONG] error condition could also be reached this way.

[ETXTBSY] System V returns this error when the executable file is currently open for writing by some process. This volume of IEEE Std 1003.1-2001 neither requires nor prohibits this behavior.

Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this volume of IEEE Std 1003.1-2001, but implementations may have a window between the call to *exec* and the time that a signal could cause one of the *exec* calls to return with [EINTR].

An explicit statement regarding the floating-point environment (as defined in the **<fenv.h>** header) was added to make it clear that the floating-point environment is set to its default when a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the default for other process and thread start-up functions is covered by more generic statements in their descriptions and can be summarized as follows:

posix_spawn( )	Set to default.
fork( )	Inherit.
pthread_create( )	Inherit.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *atexit*( ), *chmod*( ), *close*( ), *exit*( ), *fcntl*( ), *fork*( ), *fstatvfs*( ), *getenv*( ), *getitimer*( ), *getrlimit*( ), *mmap*( ), *nice*( ), *posix_spawn*( ), *posix_trace_eventid_open*( ), *posix_trace_shutdown*( ), *posix_trace_trid_eventid_open*( ), *putenv*( ), *semop*( ), *setlocale*( ), *shmat*( ), *sigaction*( ), *sigaltstack*( ), *sigpending*( ), *sigprocmask*( ), *system*( ), *times*( ), *ulimit*( ), *umask*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, behavior is defined for when the process image file is not a valid executable.
- In this issue, _POSIX_SAVED_IDS is mandated, thus the effective user ID and effective group ID of the new process image shall be saved (as the saved set-user-ID and the saved set-group-ID) for use by the *setuid*( ) function.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [EINVAL] mandatory error condition is added.
- The [ELOOP] optional error condition is added.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE PASC Interpretation 1003.1 #132 is applied.

The DESCRIPTION is updated to make it explicit that the floating-point environment in the new process image is set to the default.

The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents of a process image file affect the behavior of the *exec* functions.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change addresses a security concern, where implementations may want to reopen file descriptors 0, 1, and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of functions.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10008 **NAME**

10009 exit, _Exit, _exit — terminate a process

10010 **SYNOPSIS**

10011
```
#include <stdlib.h>
```

10012
```
void exit(int status);
```
10013
```
void _Exit(int status);
```

10014
```
#include <unistd.h>
```

10015
```
void _exit(int status);
```

10016 **DESCRIPTION**

10017 CX For *exit*( ) and *_Exit*( ): The functionality described on this reference page is aligned with the
10018 ISO C standard. Any conflict between the requirements described here and the ISO C standard is
10019 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10020 CX The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only
10021 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

10022 The *exit*( ) function shall first call all functions registered by *atexit*( ), in the reverse order of their
10023 registration, except that a function is called after any previously registered functions that had
10024 already been called at the time it was registered. Each function is called as many times as it was
10025 registered. If, during the call to any such function, a call to the *longjmp*( ) function is made that
10026 would terminate the call to the registered function, the behavior is undefined.

10027 If a function registered by a call to *atexit*( ) fails to return, the remaining registered functions shall
10028 not be called and the rest of the *exit*( ) processing shall not be completed. If *exit*( ) is called more
10029 than once, the behavior is undefined.

10030 The *exit*( ) function shall then flush all open streams with unwritten buffered data, close all open
10031 streams, and remove all files created by *tmpfile*( ). Finally, control shall be terminated with the
10032 consequences described below.

10033 CX The *_Exit*( ) and *_exit*( ) functions shall be functionally equivalent.

10034 CX The *_Exit*( ) and *_exit*( ) functions shall not call functions registered with *atexit*( ) nor any
10035 registered signal handlers. Whether open streams are flushed or closed, or temporary files are
10036 removed is implementation-defined. Finally, the calling process is terminated with the
10037 consequences described below.

10038 CX These functions shall terminate the calling process with the following consequences:

10039 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.
10040 However, XSI extensions are shaded.

10041 XSI - All of the file descriptors, directory streams, conversion descriptors, and message catalog
10042 descriptors open in the calling process shall be closed.

10043 XSI - If the parent process of the calling process is executing a *wait*( ) or *waitpid*( ), and has neither
10044 set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, it shall be notified of the calling
10045 process' termination and the low-order eight bits (that is, bits 0377) of *status* shall be made
10046 available to it. If the parent is not waiting, the child's status shall be made available to it
10047 when the parent subsequently executes *wait*( ) or *waitpid*( ).

10048 XSI The semantics of the *waitid*( ) function shall be equivalent to *wait*( ).

10049 XSI - If the parent process of the calling process is not executing a *wait*( ) or *waitpid*( ), and has
10050 neither set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, the calling process shall
10051 be transformed into a *zombie process*. A *zombie process* is an inactive process and it shall be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

deleted at some later time when its parent process executes *wait*( ) or *waitpid*( ).

XSI The semantics of the *waitid*( ) function shall be equivalent to *wait*( ).

- Termination of a process does not directly terminate its children. The sending of a SIGHUP signal as described below indirectly terminates children in some circumstances.
- Either:

  If the implementation supports the SIGCHLD signal, a SIGCHLD shall be sent to the parent process.

  Or:

  XSI If the parent process has set its SA_NOCLDWAIT flag, or set SIGCHLD to SIG_IGN, the status shall be discarded, and the lifetime of the calling process shall end immediately. If SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to the parent process.
- The parent process ID of all of the calling process' existing child processes and zombie processes shall be set to the process ID of an implementation-defined system process. That is, these processes shall be inherited by a special system process.
- XSI Each attached shared-memory segment is detached and the value of *shm_nattch* (see *shmget*( )) in the data structure associated with its shared memory ID shall be decremented by 1.
- XSI For each semaphore for which the calling process has set a *semadj* value (see *semop*( )), that value shall be added to the *semval* of the specified semaphore.
- If the process is a controlling process, the SIGHUP signal shall be sent to each process in the foreground process group of the controlling terminal belonging to the calling process.
- If the process is a controlling process, the controlling terminal associated with the session shall be disassociated from the session, allowing it to be acquired by a new controlling process.
- If the exit of the process causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal shall be sent to each process in the newly-orphaned process group.
- SEM All open named semaphores in the calling process shall be closed as if by appropriate calls to *sem_close*( ).
- ML Any memory locks established by the process via calls to *mlockall*( ) or *mlock*( ) shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to *_Exit*( ) or *_exit*( ).
- MF|SHM Memory mappings that were created in the process shall be unmapped before the process is destroyed.
- TYM Any blocks of typed memory that were mapped in the calling process shall be unmapped, as if *munmap*( ) was implicitly called to unmap them.
- MSG All open message queue descriptors in the calling process shall be closed as if by appropriate calls to *mq_close*( ).
- AIO Any outstanding cancelable asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the *_Exit*( ) or *_exit*( ) operation had not yet occurred, but any associated signal notifications shall be suppressed.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *_Exit*() or *_exit*() operation may block awaiting such I/O completion. Whether any I/O is canceled, and which I/O may be canceled upon *_Exit*() or *_exit*(), is implementation-defined.

- Threads terminated by a call to *_Exit*() or *_exit*() shall not invoke their cancellation cleanup handlers or per-thread data destructors.

TRC
- If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described by the *posix_trace_shutdown*() function, and any process' mapping of trace event names to trace event type identifiers built for these trace streams may be deallocated.

RETURN VALUE

These functions do not return.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Normally applications should use *exit*() rather than *_Exit*() or *_exit*().

RATIONALE

Process Termination

Early proposals drew a distinction between normal and abnormal process termination. Abnormal termination was caused only by certain signals and resulted in implementation-defined ''actions'', as discussed below. Subsequent proposals distinguished three types of termination: *normal termination* (as in the current specification), *simple abnormal termination*, and *abnormal termination with actions*. Again the distinction between the two types of abnormal termination was that they were caused by different signals and that implementation-defined actions would result in the latter case. Given that these actions were completely implementation-defined, the early proposals were only saying when the actions could occur and how their occurrence could be detected, but not what they were. This was of little or no use to conforming applications, and thus the distinction is not made in this volume of IEEE Std 1003.1-2001.

The implementation-defined actions usually include, in most historical implementations, the creation of a file named **core** in the current working directory of the process. This file contains an image of the memory of the process, together with descriptive information about the process, perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

There is a potential security problem in creating a **core** file if the process was set-user-ID and the current user is not the owner of the program, if the process was set-group-ID and none of the user's groups match the group of the program, or if the user does not have permission to write in the current directory. In this situation, an implementation either should not create a **core** file or should make it unreadable by the user.

Despite the silence of this volume of IEEE Std 1003.1-2001 on this feature, applications are advised not to create files named **core** because of potential conflicts in many implementations. Some implementations use a name other than **core** for the file; for example, by appending the process ID to the filename.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Terminating a Process

It is important that the consequences of process termination as described occur regardless of whether the process called *_exit*( ) (perhaps indirectly through *exit*( )) or instead was terminated due to a signal or for some other reason. Note that in the specific case of *exit*( ) this means that the *status* argument to *exit*( ) is treated in the same way as the *status* argument to *_exit*( ).

A language other than C may have other termination primitives than the C-language *exit*( ) function, and programs written in such a language should use its native termination primitives, but those should have as part of their function the behavior of *_exit*( ) as described. Implementations in languages other than C are outside the scope of this version of this volume of IEEE Std 1003.1-2001, however.

As required by the ISO C standard, using **return** from *main*( ) has the same behavior (other than with respect to language scope issues) as calling *exit*( ) with the returned value. Reaching the end of the *main*( ) function has the same behavior as calling *exit*(0).

A value of zero (or EXIT_SUCCESS, which is required to be zero) for the argument *status* conventionally indicates successful termination. This corresponds to the specification for *exit*( ) in the ISO C standard. The convention is followed by utilities such as *make* and various shells, which interpret a zero status from a child process as success. For this reason, applications should not call *exit*(0) or *_exit*(0) when they terminate unsuccessfully; for example, in signal-catching functions.

Historically, the implementation-defined process that inherits children whose parents have terminated without waiting on them is called *init* and has a process ID of 1.

The sending of a SIGHUP to the foreground process group when a controlling process terminates corresponds to somewhat different historical implementations. In System V, the kernel sends a SIGHUP on termination of (essentially) a controlling process. In 4.2 BSD, the kernel does not send SIGHUP in a case like this, but the termination of a controlling process is usually noticed by a system daemon, which arranges to send a SIGHUP to the foreground process group with the *vhangup*( ) function. However, in 4.2 BSD, due to the behavior of the shells that support job control, the controlling process is usually a shell with no other processes in its process group. Thus, a change to make *_exit*( ) behave this way in such systems should not cause problems with existing applications.

The termination of a process may cause a process group to become orphaned in either of two ways. The connection of a process group to its parent(s) outside of the group depends on both the parents and their children. Thus, a process group may be orphaned by the termination of the last connecting parent process outside of the group or by the termination of the last direct descendant of the parent process(es). In either case, if the termination of a process causes a process group to become orphaned, processes within the group are disconnected from their job control shell, which no longer has any information on the existence of the process group. Stopped processes within the group would languish forever. In order to avoid this problem, newly orphaned process groups that contain stopped processes are sent a SIGHUP signal and a SIGCONT signal to indicate that they have been disconnected from their session. The SIGHUP signal causes the process group members to terminate unless they are catching or ignoring SIGHUP. Under most circumstances, all of the members of the process group are stopped if any of them are stopped.

The action of sending a SIGHUP and a SIGCONT signal to members of a newly orphaned process group is similar to the action of 4.2 BSD, which sends SIGHUP and SIGCONT to each stopped child of an exiting process. If such children exit in response to the SIGHUP, any additional descendants receive similar treatment at that time. In this volume of IEEE Std 1003.1-2001, the signals are sent to the entire process group at the same time. Also, in

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

this volume of IEEE Std 1003.1-2001, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a process group that is not orphaned; therefore, the action taken at *_exit*( ) must consider processes other than child processes.

It is possible for a process group to be orphaned by a call to *setpgid*( ) or *setsid*( ), as well as by process termination. This volume of IEEE Std 1003.1-2001 does not require sending SIGHUP and SIGCONT in those cases, because, unlike process termination, those cases are not caused accidentally by applications that are unaware of job control. An implementation can choose to send SIGHUP and SIGCONT in those cases as an extension; such an extension must be documented as required in **<signal.h>**.

The ISO/IEC 9899:1999 standard adds the *_Exit*( ) function that results in immediate program termination without triggering signals or *atexit*( )-registered functions. In IEEE Std 1003.1-2001, this is equivalent to the *_exit*( ) function.

FUTURE DIRECTIONS

None.

SEE ALSO

atexit( ), *close*( ), *fclose*( ), *longjmp*( ), *posix_trace_shutdown*( ), *posix_trace_trid_eventid_open*( ), *semop*( ), *shmget*( ), *sigaction*( ), *wait*( ), *waitid*( ), *waitpid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are further clarified.

The values of *status* from *exit*( ) are better described.

Issue 6

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *_Exit*( ) function is included.
- The DESCRIPTION is updated.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

References to the *wait3*( ) function are removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the DESCRIPTION.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10222 **NAME**

10223 exp, expf, expl — exponential function

10224 **SYNOPSIS**

10225 `#include <math.h>`

10226 `double exp(double x);`
10227 `float expf(float x);`
10228 `long double expl(long double x);`

10229 **DESCRIPTION**

10230 CX The functionality described on this reference page is aligned with the ISO C standard. Any
10231 conflict between the requirements described here and the ISO C standard is unintentional. This
10232 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10233 These functions shall compute the base-*e* exponential of *x*.

10234 An application wishing to check for error situations should set *errno* to zero and call
10235 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
10236 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
10237 zero, an error has occurred.

10238 **RETURN VALUE**

10239 Upon successful completion, these functions shall return the exponential value of *x*.

10240 If the correct value would cause overflow, a range error shall occur and *exp*( ), *expf*( ), and *expl*( )
10241 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

10242 If the correct value would cause underflow, and is not representable, a range error may occur,
10243 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

10244 MX If *x* is NaN, a NaN shall be returned.

10245 If *x* is ±0, 1 shall be returned.

10246 If *x* is –Inf, +0 shall be returned.

10247 If *x* is +Inf, *x* shall be returned.

10248 If the correct value would cause underflow, and is representable, a range error may occur and
10249 the correct value shall be returned.

10250 **ERRORS**

10251 These functions shall fail if:

10252 Range Error The result overflows.

10253 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
10254 then *errno* shall be set to [ERANGE]. If the integer expression
10255 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
10256 floating-point exception shall be raised.

10257 These functions may fail if:

10258 Range Error The result underflows.

10259 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
10260 then *errno* shall be set to [ERANGE]. If the integer expression
10261 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
10262 floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

Note that for IEEE Std 754-1985 **double**, 709.8 < *x* implies *exp*(*x*) has overflowed. The value *x* < –708.4 implies *exp*(*x*) has underflowed.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *log*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *expf*( ) and *expl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10288 **NAME**

10289 exp2, exp2f, exp2l — exponential base 2 functions

10290 **SYNOPSIS**

10291
```
#include <math.h>
```

10292
```
double exp2(double x);
```
10293
```
float exp2f(float x);
```
10294
```
long double exp2l(long double x);
```

10295 **DESCRIPTION**

10296 CX The functionality described on this reference page is aligned with the ISO C standard. Any
10297 conflict between the requirements described here and the ISO C standard is unintentional. This
10298 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10299 These functions shall compute the base-2 exponential of *x*.

10300 An application wishing to check for error situations should set *errno* to zero and call
10301 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
10302 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
10303 zero, an error has occurred.

10304 **RETURN VALUE**

10305 Upon successful completion, these functions shall return 2^x.

10306 If the correct value would cause overflow, a range error shall occur and *exp2*( ), *exp2f*( ), and
10307 *exp2l*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
10308 respectively.

10309 If the correct value would cause underflow, and is not representable, a range error may occur,
10310 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

10311 MX If *x* is NaN, a NaN shall be returned.

10312 If *x* is ±0, 1 shall be returned.

10313 If *x* is –Inf, +0 shall be returned.

10314 If *x* is +Inf, *x* shall be returned.

10315 If the correct value would cause underflow, and is representable, a range error may occur and
10316 the correct value shall be returned.

10317 **ERRORS**

10318 These functions shall fail if:

10319 Range Error The result overflows.

10320 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
10321 then *errno* shall be set to [ERANGE]. If the integer expression
10322 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
10323 floating-point exception shall be raised.

10324 These functions may fail if:

10325 Range Error The result underflows.

10326 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
10327 then *errno* shall be set to [ERANGE]. If the integer expression
10328 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
10329 floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10330 **EXAMPLES**

10331 None.

10332 **APPLICATION USAGE**

10333 For IEEE Std 754-1985 **double**, 1024 <= *x* implies *exp2*(*x*) has overflowed. The value *x* < –1022
10334 implies *exp*(*x*) has underflowed.

10335 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
10336 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

10337 **RATIONALE**

10338 None.

10339 **FUTURE DIRECTIONS**

10340 None.

10341 **SEE ALSO**

10342 *exp*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *log*( ), the Base Definitions volume of
10343 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
10344 **<math.h>**

10345 **CHANGE HISTORY**

10346 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10347 **NAME**

10348 expm1, expm1f, expm1l — compute exponential functions

10349 **SYNOPSIS**

10350 `#include <math.h>`

10351 `double expm1(double x);`
10352 `float expm1f(float x);`
10353 `long double expm1l(long double x);`

10354 **DESCRIPTION**

10355 CX The functionality described on this reference page is aligned with the ISO C standard. Any
10356 conflict between the requirements described here and the ISO C standard is unintentional. This
10357 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10358 These functions shall compute e^x–1.0.

10359 An application wishing to check for error situations should set *errno* to zero and call
10360 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
10361 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
10362 zero, an error has occurred.

10363 **RETURN VALUE**

10364 Upon successful completion, these functions return e^x–1.0.

10365 If the correct value would cause overflow, a range error shall occur and *expm1*( ), *expm1f*( ), and
10366 *expm1l*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
10367 respectively.

10368 MX If *x* is NaN, a NaN shall be returned.

10369 If *x* is ±0, ±0 shall be returned.

10370 If *x* is –Inf, –1 shall be returned.

10371 If *x* is +Inf, *x* shall be returned.

10372 If *x* is subnormal, a range error may occur and *x* should be returned.

10373 **ERRORS**

10374 These functions shall fail if:

10375 Range Error The result overflows.

10376 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
10377 then *errno* shall be set to [ERANGE]. If the integer expression
10378 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
10379 floating-point exception shall be raised.

10380 These functions may fail if:

10381 MX Range Error The value of *x* is subnormal.

10382 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
10383 then *errno* shall be set to [ERANGE]. If the integer expression
10384 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
10385 floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The value of *expm1*(*x*) may be more accurate than *exp*(*x*)–1.0 for small values of *x*.

The *expm1*( ) and *log1p*( ) functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:

```
expm1(n * log1p(x))/x
```

when *x* is very small (for example, when calculating small daily interest rates). These functions also simplify writing accurate inverse hyperbolic functions.

For IEEE Std 754-1985 **double**, 709.8 < *x* implies *expm1*(*x*) has overflowed.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *ilogb*( ), *log1p*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *expm1f*( ) and *expm1l*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The *expm1*( ) function is no longer marked as an extension.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

fabs, fabsf, fabsl — absolute value function

SYNOPSIS

```
#include <math.h>

double fabs(double x);
float fabsf(float x);
long double fabsl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the absolute value of their argument x, $|x|$.

RETURN VALUE

Upon successful completion, these functions shall return the absolute value of x.

MX If x is NaN, a NaN shall be returned.

If x is ±0, +0 shall be returned.

If x is ±Inf, +Inf shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isnan( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *fabsf*( ) and *fabsl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fattach — attach a STREAMS-based file descriptor to a file in the file system name space (**STREAMS**)

SYNOPSIS

XSR
```
#include <stropts.h>

int fattach(int fildes, const char *path);
```

DESCRIPTION

The *fattach*( ) function shall attach a STREAMS-based file descriptor to a file, effectively associating a pathname with *fildes*. The application shall ensure that the *fildes* argument is a valid open file descriptor associated with a STREAMS file. The *path* argument points to a pathname of an existing file. The application shall have the appropriate privileges or be the owner of the file named by *path* and have write permission. A successful call to *fattach*( ) shall cause all pathnames that name the file named by *path* to name the STREAMS file associated with *fildes*, until the STREAMS file is detached from the file. A STREAMS file can be attached to more than one file and can have several pathnames associated with it.

The attributes of the named STREAMS file shall be initialized as follows: the permissions, user ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1, and the size and device identifier are set to those of the STREAMS file associated with *fildes*. If any attributes of the named STREAMS file are subsequently changed (for example, by *chmod*( )), neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildes* refers shall be affected.

File descriptors referring to the underlying file, opened prior to an *fattach*( ) call, shall continue to refer to the underlying file.

RETURN VALUE

Upon successful completion, *fattach*( ) shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fattach*( ) function shall fail if:

[EACCES] Search permission is denied for a component of the path prefix, or the process is the owner of *path* but does not have write permissions on the file named by *path*.

[EBADF] The *fildes* argument is not a valid open file descriptor.

[EBUSY] The file named by *path* is currently a mount point or has a STREAMS file attached to it.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The size of *path* exceeds {PATH_MAX} or a component of *path* is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

[EPERM] The effective user ID of the process is not the owner of the file named by *path* and the process does not have appropriate privilege.

--`,,,`--`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *fattach*( ) function may fail if:

[EINVAL] The *fildes* argument does not refer to a STREAMS file.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[EXDEV] A link to a file on another file system was attempted.

EXAMPLES

Attaching a File Descriptor to a File

In the following example, *fd* refers to an open STREAMS file. The call to *fattach*( ) associates this STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-STREAM**, prior to breaking the attachment via a call to *fdetach*( ), will instead create a new file handle referring to the STREAMS file associated with *fd*.

```
#include <stropts.h>
...
    int fd;
    char *filename = "/tmp/named-STREAM";
    int ret;

    ret = fattach(fd, filename);
```

APPLICATION USAGE

The *fattach*( ) function behaves similarly to the traditional *mount*( ) function in the way a file is temporarily replaced by the root directory of the mounted file system. In the case of *fattach*( ), the replaced file need not be a directory and the replacing file is a STREAMS file.

RATIONALE

The file attributes of a file which has been the subject of an *fattach*( ) call are specifically set because of an artefact of the original implementation. The internal mechanism was the same as for the *mount*( ) function. Since *mount*( ) is typically only applied to directories, the effects when applied to a regular file are a little surprising, especially as regards the link count which rigidly remains one, even if there were several links originally and despite the fact that all original links refer to the STREAM as long as the *fattach*( ) remains in effect.

FUTURE DIRECTIONS

None.

SEE ALSO

fdetach( ), *isastream*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stropts.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The [EXDEV] error is added to the list of optional errors in the ERRORS section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10541 **Issue 6**

10542 This function is marked as part of the XSI STREAMS Option Group.

10543 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

10544 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
10545 [ELOOP] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10546 **NAME**

10547 fchdir — change working directory

10548 **SYNOPSIS**

10549 XSI
```
#include <unistd.h>

int fchdir(int fildes);
```
10550

10551

10552 **DESCRIPTION**

10553 The *fchdir*( ) function shall be equivalent to *chdir*( ) except that the directory that is to be the new
10554 current working directory is specified by the file descriptor *fildes*.

10555 A conforming application can obtain a file descriptor for a file of type directory using *open*( ),
10556 provided that the file status flags and access modes do not contain O_WRONLY or O_RDWR.

10557 **RETURN VALUE**

10558 Upon successful completion, *fchdir*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to
10559 indicate the error. On failure the current working directory shall remain unchanged.

10560 **ERRORS**

10561 The *fchdir*( ) function shall fail if:

10562 [EACCES] Search permission is denied for the directory referenced by *fildes*.

10563 [EBADF] The *fildes* argument is not an open file descriptor.

10564 [ENOTDIR] The open file descriptor *fildes* does not refer to a directory.

10565 The *fchdir*( ) may fail if:

10566 [EINTR] A signal was caught during the execution of *fchdir*( ).

10567 [EIO] An I/O error occurred while reading from or writing to the file system.

10568 **EXAMPLES**

10569 None.

10570 **APPLICATION USAGE**

10571 None.

10572 **RATIONALE**

10573 None.

10574 **FUTURE DIRECTIONS**

10575 None.

10576 **SEE ALSO**

10577 *chdir*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

10578 **CHANGE HISTORY**

10579 First released in Issue 4, Version 2.

10580 **Issue 5**

10581 Moved from X/OPEN UNIX extension to BASE.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fchmod — change mode of a file

SYNOPSIS

```
#include <sys/stat.h>

int fchmod(int fildes, mode_t mode);
```

DESCRIPTION

The *fchmod*( ) function shall be equivalent to *chmod*( ) except that the file whose permissions are changed is specified by the file descriptor *fildes*.

SHM If *fildes* references a shared memory object, the *fchmod*( ) function need only affect the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.

TYM If *fildes* references a typed memory object, the behavior of *fchmod*( ) is unspecified.

If *fildes* refers to a socket, the behavior of *fchmod*( ) is unspecified.

XSR If *fildes* refers to a STREAM (which is *fattach*( )-ed into the file system name space) the call returns successfully, doing nothing.

RETURN VALUE

Upon successful completion, *fchmod*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *fchmod*( ) function shall fail if:

[EBADF] The *fildes* argument is not an open file descriptor.

[EPERM] The effective user ID does not match the owner of the file and the process does not have appropriate privilege.

[EROFS] The file referred to by *fildes* resides on a read-only file system.

The *fchmod*( ) function may fail if:

XSI [EINTR] The *fchmod*( ) function was interrupted by a signal.

XSI [EINVAL] The value of the *mode* argument is invalid.

[EINVAL] The *fildes* argument refers to a pipe and the implementation disallows execution of *fchmod*( ) on a pipe.

EXAMPLES

Changing the Current Permissions for a File

The following example shows how to change the permissions for a file named **/home/cnd/mod1** so that the owner and group have read/write/execute permissions, but the world only has read/write permissions.

```
#include <sys/stat.h>
#include <fcntl.h>

mode_t mode;
int    fildes;
...
fildes = open("/home/cnd/mod1", O_RDWR);
fchmod(fildes, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *chown*( ), *creat*( ), *fcntl*( ), *fstatvfs*( ), *mknod*( ), *open*( ), *read*( ), *stat*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod*( ) in the POSIX Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a second instance of [EINVAL] is defined in the list of optional errors.

Issue 6

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod*( ) behavior is unspecified for typed memory objects.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fchown — change owner and group of a file

SYNOPSIS

```
#include <unistd.h>

int fchown(int fildes, uid_t owner, gid_t group);
```

DESCRIPTION

The *fchown*( ) function shall be equivalent to *chown*( ) except that the file whose owner and group are changed is specified by the file descriptor *fildes*.

RETURN VALUE

Upon successful completion, *fchown*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *fchown*( ) function shall fail if:

[EBADF] The *fildes* argument is not an open file descriptor.

[EPERM] The effective user ID does not match the owner of the file or the process does not have appropriate privilege and _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

[EROFS] The file referred to by *fildes* resides on a read-only file system.

The *fchown*( ) function may fail if:

[EINVAL] The owner or group ID is not a value supported by the implementation. The *fildes* argument refers to a pipe or socket or an *fattach*( )-ed STREAM and the implementation disallows execution of *fchown*( ) on a pipe. XSR

[EIO] A physical I/O error has occurred.

[EINTR] The *fchown*( ) function was interrupted by a signal which was caught.

EXAMPLES

Changing the Current Owner of a File

The following example shows how to change the owner of a file named **/home/cnd/mod1** to ‘‘jones’’ and the group to ‘‘cnd’’.

The numeric value for the user ID is obtained by extracting the user ID from the user database entry associated with ‘‘jones’’. Similarly, the numeric value for the group ID is obtained by extracting the group ID from the group database entry associated with ‘‘cnd’’. This example assumes the calling program has appropriate privileges.

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <pwd.h>
#include <grp.h>

struct passwd *pwd;
struct group  *grp;
int            fildes;
...
fildes = open("/home/cnd/mod1", O_RDWR);
pwd = getpwnam("jones");
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
        grp = getgrnam("cnd");
        fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that a call to *fchown*( ) may not be allowed on a pipe.

The *fchown*( ) function is defined as mandatory.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fclose — close a stream

SYNOPSIS

```
#include <stdio.h>

int fclose(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fclose*( ) function shall cause the stream pointed to by *stream* to be flushed and the associated file to be closed. Any unwritten buffered data for the stream shall be written to the file; any unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be disassociated from the file and any buffer set by the *setbuf*( ) or *setvbuf*( ) function shall be disassociated from the stream. If the associated buffer was automatically allocated, it shall be deallocated.

CX The *fclose*( ) function shall mark for update the *st_ctime* and *st_mtime* fields of the underlying file, if the stream was writable, and if buffered data remains that has not yet been written to the file. The *fclose*( ) function shall perform the equivalent of a *close*( ) on the file descriptor that is associated with the stream pointed to by *stream*.

After the call to *fclose*( ), any use of *stream* results in undefined behavior.

RETURN VALUE

CX Upon successful completion, *fclose*( ) shall return 0; otherwise, it shall return EOF and set *errno* to indicate the error.

ERRORS

The *fclose*( ) function shall fail if:

CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the process would be delayed in the write operation.

CX [EBADF] The file descriptor underlying stream is not valid.

CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

CX [EINTR] The *fclose*( ) function was interrupted by a signal.

CX [EIO] The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

CX [ENOSPC] There was no free space remaining on the device containing the file.

CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fclose*( ) function may fail if:

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *fopen*( ), *getrlimit*( ), *ulimit*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] error is added as part of the large file support extensions.
- The [ENXIO] optional error condition is added.

The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10767 **NAME**

10768 fcntl — file control

10769 **SYNOPSIS**

10770 OH
```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fildes, int cmd, ...);
```

10773 **DESCRIPTION**

10774 The *fcntl*( ) function shall perform the operations described below on open files. The *fildes*
10775 argument is a file descriptor.

10776 The available values for *cmd* are defined in **<fcntl.h>** and are as follows:

10777 F_DUPFD Return a new file descriptor which shall be the lowest numbered available
10778 (that is, not already open) file descriptor greater than or equal to the third
10779 argument, *arg*, taken as an integer of type **int**. The new file descriptor shall
10780 refer to the same open file description as the original file descriptor, and shall
10781 share any locks. The FD_CLOEXEC flag associated with the new file
10782 descriptor shall be cleared to keep the file open across calls to one of the *exec*
10783 functions.

10784 F_GETFD Get the file descriptor flags defined in **<fcntl.h>** that are associated with the
10785 file descriptor *fildes*. File descriptor flags are associated with a single file
10786 descriptor and do not affect other file descriptors that refer to the same file.

10787 F_SETFD Set the file descriptor flags defined in **<fcntl.h>**, that are associated with *fildes*,
10788 to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC flag in the
10789 third argument is 0, the file shall remain open across the *exec* functions;
10790 otherwise, the file shall be closed upon successful execution of one of the *exec*
10791 functions.

10792 F_GETFL Get the file status flags and file access modes, defined in **<fcntl.h>**, for the file
10793 description associated with *fildes*. The file access modes can be extracted from
10794 the return value using the mask O_ACCMODE, which is defined in **<fcntl.h>**.
10795 File status flags and file access modes are associated with the file description
10796 and do not affect other file descriptors that refer to the same file with different
10797 open file descriptions.

10798 F_SETFL Set the file status flags, defined in **<fcntl.h>**, for the file description associated
10799 with *fildes* from the corresponding bits in the third argument, *arg*, taken as
10800 type **int**. Bits corresponding to the file access mode and the file creation flags,
10801 as defined in **<fcntl.h>**, that are set in *arg* shall be ignored. If any bits in *arg*
10802 other than those mentioned here are changed by the application, the result is
10803 unspecified.

10804 F_GETOWN If *fildes* refers to a socket, get the process or process group ID specified to
10805 receive SIGURG signals when out-of-band data is available. Positive values
10806 indicate a process ID; negative values, other than –1, indicate a process group
10807 ID. If *fildes* does not refer to a socket, the results are unspecified.

10808 F_SETOWN If *fildes* refers to a socket, set the process or process group ID specified to
10809 receive SIGURG signals when out-of-band data is available, using the value of
10810 the third argument, *arg*, taken as type **int**. Positive values indicate a process
10811 ID; negative values, other than –1, indicate a process group ID. If *fildes* does
10812 not refer to a socket, the results are unspecified.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The following values for *cmd* are available for advisory record locking. Record locking shall be supported for regular files, and may be supported for other files.

F_GETLK Get the first lock which blocks the lock description pointed to by the third argument, *arg*, taken as a pointer to type **struct flock**, defined in **<fcntl.h>**. The information retrieved shall overwrite the information passed to *fcntl*( ) in the structure **flock**. If no lock is found that would prevent this lock from being created, then the structure shall be left unchanged except for the lock type which shall be set to F_UNLCK.

F_SETLK Set or clear a file segment lock according to the lock description pointed to by the third argument, *arg*, taken as a pointer to type **struct flock**, defined in **<fcntl.h>**. F_SETLK can establish shared (or read) locks (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in **<fcntl.h>**. If a shared or exclusive lock cannot be set, *fcntl*( ) shall return immediately with a return value of −1.

F_SETLKW This command shall be equivalent to F_SETLK except that if a shared or exclusive lock is blocked by other locks, the thread shall wait until the request can be satisfied. If a signal that is to be caught is received while *fcntl*( ) is waiting for a region, *fcntl*( ) shall be interrupted. Upon return from the signal handler, *fcntl*( ) shall return −1 with *errno* set to [EINTR], and the lock operation shall not be done.

Additional implementation-defined values for *cmd* may be defined in **<fcntl.h>**. Their names shall start with F_.

When a shared lock is set on a segment of a file, other processes shall be able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the file descriptor was not opened with read access.

An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock shall fail if the file descriptor was not opened with write access.

The structure **flock** describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*), size (*l_len*), and process ID (*l_pid*) of the segment of the file to be affected.

The value of *l_whence* is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative offset *l_start* bytes shall be measured from the start of the file, current position, or end of the file, respectively. The value of *l_len* is the number of consecutive bytes to be locked. The value of *l_len* may be negative (where the definition of **off_t** permits negative values of *l_len*). The *l_pid* field is only used with F_GETLK to return the process ID of the process holding a blocking lock. After a successful F_GETLK request, when a blocking lock is found, the values returned in the **flock** structure shall be as follows:

l_type Type of blocking lock found.

l_whence SEEK_SET.

l_start Start of the blocking lock.

l_len Length of the blocking lock.

l_pid Process ID of the process that holds the blocking lock.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`--`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the command is F_SETLKW and the process must wait for another process to release a lock, then the range of bytes to be locked shall be determined before the *fcntl*( ) function blocks. If the file size or file descriptor seek offset change while *fcntl*( ) is blocked, this shall not affect the range of bytes locked.

If *l_len* is positive, the area affected shall start at *l_start* and end at *l_start*+*l_len*−1. If *l_len* is negative, the area affected shall start at *l_start*+*l_len* and end at *l_start*−1. Locks may start and extend beyond the current end of a file, but shall not extend before the beginning of the file. A lock shall be set to extend to the largest possible value of the file offset for that file by setting *l_len* to 0. If such a lock also has *l_start* set to 0 and *l_whence* is set to SEEK_SET, the whole file shall be locked.

There shall be at most one type of lock set for each byte in the file. Before a successful return from an F_SETLK or an F_SETLKW request when the calling process has previously existing locks on bytes in the region specified by the request, the previous lock type for each byte in the specified region shall be replaced by the new lock type. As specified above under the descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request (respectively) shall fail or block when another process has existing locks on bytes in the specified region and the type of any of those locks conflicts with the type specified in the request.

All locks associated with a file for a given process shall be removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process.

A potential for deadlock occurs if a process controlling a locked region is put to sleep by attempting to lock another process' locked region. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, *fcntl*( ) shall fail with an [EDEADLK] error.

An unlock (F_UNLCK) request in which *l_len* is non-zero and the offset of the last byte of the requested segment is the maximum value for an object of type **off_t**, when the process has an existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0. Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.

SHM When the file descriptor *fildes* refers to a shared memory object, the behavior of *fcntl*( ) shall be the same as for a regular file except the effect of the following values for the argument *cmd* shall be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.

TYM If *fildes* refers to a typed memory object, the result of the *fcntl*( ) function is unspecified.

RETURN VALUE

Upon successful completion, the value returned shall depend on *cmd* as follows:

F_DUPFD	A new file descriptor.
F_GETFD	Value of flags defined in **<fcntl.h>**. The return value shall not be negative.
F_SETFD	Value other than −1.
F_GETFL	Value of file status flags and access modes. The return value is not negative.
F_SETFL	Value other than −1.
F_GETLK	Value other than −1.
F_SETLK	Value other than −1.
F_SETLKW	Value other than −1.
F_GETOWN	Value of the socket owner process or process group; this will not be −1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

F_SETOWN Value other than –1.

Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fcntl*( ) function shall fail if:

[EACCES] or [EAGAIN]
: The *cmd* argument is F_SETLK; the type of lock (*l_type*) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.

[EBADF] The *fildes* argument is not a valid open file descriptor, or the argument *cmd* is F_SETLK or F_SETLKW, the type of lock, *l_type*, is a shared lock (F_RDLCK), and *fildes* is not a valid file descriptor open for reading, or the type of lock, *l_type*, is an exclusive lock (F_WRLCK), and *fildes* is not a valid file descriptor open for writing.

[EINTR] The *cmd* argument is F_SETLKW and the function was interrupted by a signal.

[EINVAL] The *cmd* argument is invalid, or the *cmd* argument is F_DUPFD and *arg* is negative or greater than or equal to {OPEN_MAX}, or the *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW and the data pointed to by *arg* is not valid, or *fildes* refers to a file that does not support locking.

[EMFILE] The argument *cmd* is F_DUPFD and {OPEN_MAX} file descriptors are currently open in the calling process, or no file descriptors greater than or equal to *arg* are available.

[ENOLCK] The argument *cmd* is F_SETLK or F_SETLKW and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.

[EOVERFLOW] One of the values to be returned cannot be represented correctly.

[EOVERFLOW] The *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or, if *l_len* is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type **off_t**.

The *fcntl*( ) function may fail if:

[EDEADLK] The *cmd* argument is F_SETLKW, the lock is blocked by a lock from another process, and putting the calling process to sleep to wait for that lock to become free would cause a deadlock.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number of arguments. It is used because System V uses pointers for the implementation of file locking functions.

The *arg* values to F_GETFD, F_SETFD, F_GETFL, and F_SETFL all represent flag values to allow for future growth. Applications using these functions should do a read-modify-write operation

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

on them, rather than assuming that only the values defined by this volume of IEEE Std 1003.1-2001 are valid. It is a common error to forget this, particularly in the case of F_SETFD.

This volume of IEEE Std 1003.1-2001 permits concurrent read and write access to file data using the *fcntl*( ) function; this is a change from the 1984 /usr/group standard and early proposals. Without concurrency controls, this feature may not be fully utilized without occasional loss of data.

Data losses occur in several ways. One case occurs when several processes try to update the same record, without sequencing controls; several updates may occur in parallel and the last writer ''wins''. Another case is a bit-tree or other internal list-based database that is undergoing reorganization. Without exclusive use to the tree segment by the updating process, other reading processes chance getting lost in the database when the index blocks are split, condensed, inserted, or deleted. While *fcntl*( ) is useful for many applications, it is not intended to be overly general and does not handle the bit-tree example well.

This facility is only required for regular files because it is not appropriate for many devices such as terminals and network connections.

Since *fcntl*( ) works with ''any file descriptor associated with that file, however it is obtained'', the file descriptor may have been inherited through a *fork*( ) or *exec* operation and thus may affect a file that another process also has open.

The use of the open file description to identify what to lock requires extra calls and presents problems if several processes are sharing an open file description, but there are too many implementations of the existing mechanism for this volume of IEEE Std 1003.1-2001 to use different specifications.

Another consequence of this model is that closing any file descriptor for a given file (whether or not it is the same open file description that created the lock) causes the locks on that file to be relinquished for that process. Equivalently, any close for any file/process pair relinquishes the locks owned on that file for that process. But note that while an open file description may be shared through *fork*( ), locks are not inherited through *fork*( ). Yet locks may be inherited through one of the *exec* functions.

The identification of a machine in a network environment is outside the scope of this volume of IEEE Std 1003.1-2001. Thus, an *l_sysid* member, such as found in System V, is not included in the locking structure.

Changing of lock types can result in a previously locked region being split into smaller regions.

Mandatory locking was a major feature of the 1984 /usr/group standard.

For advisory file record locking to be effective, all processes that have access to a file must cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode record locking is important when it cannot be assumed that all processes are cooperating. For example, if one user uses an editor to update a file at the same time that a second user executes another process that updates the same file and if only one of the two processes is using advisory locking, the processes are not cooperating. Enforcement-mode record locking would protect against accidental collisions.

Secondly, advisory record locking requires a process using locking to bracket each I/O operation with lock (or test) and unlock operations. With enforcement-mode file and record locking, a process can lock the file once and unlock when all I/O operations have been completed. Enforcement-mode record locking provides a base that can be enhanced; for example, with sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so other processes could read it, but none of them could write it.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Mandatory locks were omitted for several reasons:

1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most implementations; this was confusing, at best.
2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
3. Any publicly readable file could be locked by anyone. Many historical implementations keep the password database in a publicly readable file. A malicious user could thus prohibit logins. Another possibility would be to hold open a long-distance telephone line.
4. Some demand-paged historical implementations offer memory mapped files, and enforcement cannot be done on that type of file.

Since sleeping on a region is interrupted with any signal, *alarm*( ) may be used to provide a timeout facility in applications requiring it. This is useful in deadlock detection. Since implementation of full deadlock detection is not always feasible, the [EDEADLK] error was made optional.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *close*( ), *exec*, *open*( ), *sigaction*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<signal.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is mandated.
- In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd* is invalid, and two [EOVERFLOW] error conditions are added.

The F_GETOWN and F_SETOWN values are added for sockets.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that the extent of the bytes locked is determined prior to the blocking action.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *fcntl*( ) results are unspecified for typed memory objects.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11033 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11034 **NAME**

11035 fcvt — convert a floating-point number to a string (**LEGACY**)

11036 **SYNOPSIS**

11037 XSI
```
#include <stdlib.h>

char *fcvt(double value, int ndigit, int *restrict decpt,
    int *restrict sign);
```
11038
11039
11040

11041 **DESCRIPTION**

11042 Refer to *ecvt*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11043 **NAME**

11044 fdatasync — synchronize the data of a file (**REALTIME**)

11045 **SYNOPSIS**

11046 SIO
```
#include <unistd.h>

int fdatasync(int fildes);
```
11047

11048

11049 **DESCRIPTION**

11050 The *fdatasync*( ) function shall force all currently queued I/O operations associated with the file
11051 indicated by file descriptor *fildes* to the synchronized I/O completion state.

11052 The functionality shall be equivalent to *fsync*( ) with the symbol _POSIX_SYNCHRONIZED_IO
11053 defined, with the exception that all I/O operations shall be completed as defined for
11054 synchronized I/O data integrity completion.

11055 **RETURN VALUE**

11056 If successful, the *fdatasync*( ) function shall return the value 0; otherwise, the function shall return
11057 the value –1 and set *errno* to indicate the error. If the *fdatasync*( ) function fails, outstanding I/O
11058 operations are not guaranteed to have been completed.

11059 **ERRORS**

11060 The *fdatasync*( ) function shall fail if:

11061 [EBADF] The *fildes* argument is not a valid file descriptor open for writing.

11062 [EINVAL] This implementation does not support synchronized I/O for this file.

11063 In the event that any of the queued I/O operations fail, *fdatasync*( ) shall return the error
11064 conditions defined for *read*( ) and *write*( ).

11065 **EXAMPLES**

11066 None.

11067 **APPLICATION USAGE**

11068 None.

11069 **RATIONALE**

11070 None.

11071 **FUTURE DIRECTIONS**

11072 None.

11073 **SEE ALSO**

11074 *aio_fsync*( ), *fcntl*( ), *fsync*( ), *open*( ), *read*( ), *write*( ), the Base Definitions volume of
11075 IEEE Std 1003.1-2001, **<unistd.h>**

11076 **CHANGE HISTORY**

11077 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11078 **Issue 6**

11079 The [ENOSYS] error condition has been removed as stubs need not be provided if an
11080 implementation does not support the Synchronized Input and Output option.

11081 The *fdatasync*( ) function is marked as part of the Synchronized Input and Output option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11082 **NAME**

11083 fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)

11084 **SYNOPSIS**

11085 XSR
```
#include <stropts.h>
```

11086
```
int fdetach(const char *path);
```

11087

11088 **DESCRIPTION**

11089 The *fdetach*( ) function shall detach a STREAMS-based file from the file to which it was attached
11090 by a previous call to *fattach*( ). The *path* argument points to the pathname of the attached
11091 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A
11092 successful call to *fdetach*( ) shall cause all pathnames that named the attached STREAMS file to
11093 again name the file to which the STREAMS file was attached. All subsequent operations on *path*
11094 shall operate on the underlying file and not on the STREAMS file.

11095 All open file descriptions established while the STREAMS file was attached to the file referenced
11096 by *path* shall still refer to the STREAMS file after the *fdetach*( ) has taken effect.

11097 If there are no open file descriptors or other references to the STREAMS file, then a successful
11098 call to *fdetach*( ) shall be equivalent to performing the last *close*( ) on the attached file.

11099 **RETURN VALUE**

11100 Upon successful completion, *fdetach*( ) shall return 0; otherwise, it shall return –1 and set *errno* to
11101 indicate the error.

11102 **ERRORS**

11103 The *fdetach*( ) function shall fail if:

11104 [EACCES] Search permission is denied on a component of the path prefix.

11105 [EINVAL] The *path* argument names a file that is not currently attached.

11106 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
11107 argument.

11108 [ENAMETOOLONG]
11109 The size of a pathname exceeds {PATH_MAX} or a pathname component is
11110 longer than {NAME_MAX}.

11111 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

11112 [ENOTDIR] A component of the path prefix is not a directory.

11113 [EPERM] The effective user ID is not the owner of *path* and the process does not have
11114 appropriate privileges.

11115 The *fdetach*( ) function may fail if:

11116 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
11117 resolution of the *path* argument.

11118 [ENAMETOOLONG]
11119 Pathname resolution of a symbolic link produced an intermediate result
11120 whose length exceeds {PATH_MAX}.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Detaching a File

The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to which it was attached by a previous, successful call to *fattach*( ). Subsequent calls to open this file refer to the underlying file, not to the STREAMS file.

```
#include <stropts.h>
...
    char *filename = "/tmp/named-STREAM";
    int ret;

    ret = fdetach(filename);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fattach( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stropts.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

fdim, fdimf, fdiml — compute positive difference between two floating-point numbers

SYNOPSIS

```
#include <math.h>

double fdim(double x, double y);
float fdimf(float x, float y);
long double fdiml(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall determine the positive difference between their arguments. If *x* is greater than *y*, *x*–*y* is returned. If *x* is less than or equal to *y*, +0 is returned.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the positive difference value.

If *x*–*y* is positive and overflows, a range error shall occur and *fdim*( ), *fdimf*( ), and *fdiml*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

XSI If *x*–*y* is positive and underflows, a range error may occur, and either (*x*–*y*) (if representable), or 0.0 (if supported), or an implementation-defined value shall be returned.

MX If *x* or *y* is NaN, a NaN shall be returned.

ERRORS

The *fdim*( ) function shall fail if:

Range Error — The result overflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

The *fdim*( ) function may fail if:

Range Error — The result underflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On implementations supporting IEEE Std 754-1985, $x-y$ cannot underflow, and hence the 0.0 return value is shaded as an extension for implementations supporting the XSI extension rather than an MX extension.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *fmax*( ), *fmin*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

fdopen — associate a stream with a file descriptor

SYNOPSIS

CX
```
#include <stdio.h>

FILE *fdopen(int fildes, const char *mode);
```

DESCRIPTION

The *fdopen*( ) function shall associate a stream with a file descriptor.

The *mode* argument is a character string having one of the following values:

r or *rb*	Open a file for reading.
w or *wb*	Open a file for writing.
a or *ab*	Open a file for writing at end-of-file.
r+ or *rb+* or *r+b*	Open a file for update (reading and writing).
w+ or *wb+* or *w+b*	Open a file for update (reading and writing).
a+ or *ab+* or *a+b*	Open a file for update (reading and writing) at end-of-file.

The meaning of these flags is exactly as specified in *fopen*( ), except that modes beginning with *w* shall not cause truncation of the file.

Additional values for the *mode* argument may be supported by an implementation.

The application shall ensure that the mode of the stream as expressed by the *mode* argument is allowed by the file access mode of the open file description to which *fildes* refers. The file position indicator associated with the new stream is set to the position indicated by the file offset associated with the file descriptor.

The error and end-of-file indicators for the stream shall be cleared. The *fdopen*( ) function may cause the *st_atime* field of the underlying file to be marked for update.

SHM If *fildes* refers to a shared memory object, the result of the *fdopen*( ) function is unspecified.

TYM If *fildes* refers to a typed memory object, the result of the *fdopen*( ) function is unspecified.

The *fdopen*( ) function shall preserve the offset maximum previously set for the open file description corresponding to *fildes*.

RETURN VALUE

Upon successful completion, *fdopen*( ) shall return a pointer to a stream; otherwise, a null pointer shall be returned and *errno* set to indicate the error.

ERRORS

The *fdopen*( ) function may fail if:

[EBADF]	The *fildes* argument is not a valid file descriptor.
[EINVAL]	The *mode* argument is not a valid mode.
[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.
[EMFILE]	{STREAM_MAX} streams are currently open in the calling process.
[ENOMEM]	Insufficient space to allocate a buffer.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

File descriptors are obtained from calls like *open*( ), *dup*( ), *creat*( ), or *pipe*( ), which open files but do not return streams.

RATIONALE

The file descriptor may have been obtained from *open*( ), *creat*( ), *pipe*( ), *dup*( ), or *fcntl*( ); inherited through *fork*( ) or *exec*; or perhaps obtained by implementation-defined means, such as the 4.3 BSD *socket*( ) call.

The meanings of the *mode* arguments of *fdopen*( ) and *fopen*( ) differ. With *fdopen*( ), open for write (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument formats that include *a b* are allowed for consistency with the ISO C standard function *fopen*( ). The *b* has no effect on the resulting stream. Although not explicitly required by this volume of IEEE Std 1003.1-2001, a good implementation of append (*a*) mode would cause the O_APPEND flag to be set.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5.1 (on page 35), *fclose*( ), *fopen*( ), *open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the use and setting of the *mode* argument are changed to include binary streams.
- In the DESCRIPTION, text is added for large file support to indicate setting of the offset maximum in the open file description.
- All errors identified in the ERRORS section are added.
- In the DESCRIPTION, text is added that the *fdopen*( ) function may cause *st_atime* to be updated.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that it is the responsibility of the application to ensure that the mode is compatible with the open file descriptor.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *fdopen*( ) results are unspecified for typed memory objects.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11279 **NAME**

11280 feclearexcept — clear floating-point exception

11281 **SYNOPSIS**

11282 `#include <fenv.h>`

11283 `int feclearexcept(int excepts);`

11284 **DESCRIPTION**

11285 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11286 conflict between the requirements described here and the ISO C standard is unintentional. This
11287 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11288 The *feclearexcept*( ) function shall attempt to clear the supported floating-point exceptions
11289 represented by *excepts*.

11290 **RETURN VALUE**

11291 If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept*( )
11292 shall return zero. Otherwise, it shall return a non-zero value.

11293 **ERRORS**

11294 No errors are defined.

11295 **EXAMPLES**

11296 None.

11297 **APPLICATION USAGE**

11298 None.

11299 **RATIONALE**

11300 None.

11301 **FUTURE DIRECTIONS**

11302 None.

11303 **SEE ALSO**

11304 *fegetexceptflag*( ), *feraiseexcept*( ), *fesetexceptflag*( ), *fetestexcept*( ), the Base Definitions volume of
11305 IEEE Std 1003.1-2001, **<fenv.h>**

11306 **CHANGE HISTORY**

11307 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

11308 ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11309 **NAME**

11310 fegetenv, fesetenv — get and set current floating-point environment

11311 **SYNOPSIS**

11312
```
#include <fenv.h>
```

11313
```
int fegetenv(fenv_t *envp);
```
11314
```
int fesetenv(const fenv_t *envp);
```

11315 **DESCRIPTION**

11316 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11317 conflict between the requirements described here and the ISO C standard is unintentional. This
11318 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11319 The *fegetenv*( ) function shall attempt to store the current floating-point environment in the object
11320 pointed to by *envp*.

11321 The *fesetenv*( ) function shall attempt to establish the floating-point environment represented by
11322 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to
11323 *fegetenv*( ) or *feholdexcept*( ), or equal a floating-point environment macro. The *fesetenv*( ) function
11324 does not raise floating-point exceptions, but only installs the state of the floating-point status
11325 flags represented through its argument.

11326 **RETURN VALUE**

11327 If the representation was successfully stored, *fegetenv*( ) shall return zero. Otherwise, it shall
11328 return a non-zero value. If the environment was successfully established, *fesetenv*( ) shall return
11329 zero. Otherwise, it shall return a non-zero value.

11330 **ERRORS**

11331 No errors are defined.

11332 **EXAMPLES**

11333 None.

11334 **APPLICATION USAGE**

11335 None.

11336 **RATIONALE**

11337 None.

11338 **FUTURE DIRECTIONS**

11339 None.

11340 **SEE ALSO**

11341 *feholdexcept*( ), *feupdateenv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fenv.h>**

11342 **CHANGE HISTORY**

11343 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

11344 ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fegetexceptflag, fesetexceptflag — get and set floating-point status flags

SYNOPSIS

```
#include <fenv.h>

int fegetexceptflag(fexcept_t *flagp, int excepts);
int fesetexceptflag(const fexcept_t *flagp, int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fegetexceptflag*( ) function shall attempt to store an implementation-defined representation of the states of the floating-point status flags indicated by the argument *excepts* in the object pointed to by the argument *flagp*.

The *fesetexceptflag*( ) function shall attempt to set the floating-point status flags indicated by the argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by *flagp* shall have been set by a previous call to *fegetexceptflag*( ) whose second argument represented at least those floating-point exceptions represented by the argument *excepts*. This function does not raise floating-point exceptions, but only sets the state of the flags.

RETURN VALUE

If the representation was successfully stored, *fegetexceptflag*( ) shall return zero. Otherwise, it shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions were successfully set, *fesetexceptflag*( ) shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *feraiseexcept*( ), *fetestexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fegetround, fesetround — get and set current rounding direction

SYNOPSIS

```
#include <fenv.h>

int fegetround(void);
int fesetround(int round);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fegetround*( ) function shall get the current rounding direction.

The *fesetround*( ) function shall establish the rounding direction represented by its argument *round*. If the argument is not equal to the value of a rounding direction macro, the rounding direction is not changed.

RETURN VALUE

The *fegetround*( ) function shall return the value of the rounding direction macro representing the current rounding direction or a negative value if there is no such rounding direction macro or the current rounding direction is not determinable.

The *fesetround*( ) function shall return a zero value if and only if the requested rounding direction was established.

ERRORS

No errors are defined.

EXAMPLES

The following example saves, sets, and restores the rounding direction, reporting an error and aborting if setting the rounding direction fails:

```
#include <fenv.h>
#include <assert.h>
void f(int round_dir)
{
    #pragma STDC FENV_ACCESS ON
    int save_round;
    int setround_ok;
    save_round = fegetround();
    setround_ok = fesetround(round_dir);
    assert(setround_ok == 0);
    /* ... */
    fesetround(save_round);
    /* ... */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11427 **FUTURE DIRECTIONS**

11428 None.

11429 **SEE ALSO**

11430 The Base Definitions volume of IEEE Std 1003.1-2001, **<fenv.h>**

11431 **CHANGE HISTORY**

11432 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

11433 ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11434 **NAME**

11435 feholdexcept — save current floating-point environment

11436 **SYNOPSIS**

11437 `#include <fenv.h>`

11438 `int feholdexcept(fenv_t *envp);`

11439 **DESCRIPTION**

11440 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11441 conflict between the requirements described here and the ISO C standard is unintentional. This
11442 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11443 The *feholdexcept*( ) function shall save the current floating-point environment in the object
11444 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on
11445 floating-point exceptions) mode, if available, for all floating-point exceptions.

11446 **RETURN VALUE**

11447 The *feholdexcept*( ) function shall return zero if and only if non-stop floating-point exception
11448 handling was successfully installed.

11449 **ERRORS**

11450 No errors are defined.

11451 **EXAMPLES**

11452 None.

11453 **APPLICATION USAGE**

11454 None.

11455 **RATIONALE**

11456 The *feholdexcept*( ) function should be effective on typical IEC 60559:1989 standard
11457 implementations which have the default non-stop mode and at least one other mode for trap
11458 handling or aborting. If the implementation provides only the non-stop mode, then installing the
11459 non-stop mode is trivial.

11460 **FUTURE DIRECTIONS**

11461 None.

11462 **SEE ALSO**

11463 *fegetenv*( ), *fesetenv*( ), *feupdateenv*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
11464 **<fenv.h>**

11465 **CHANGE HISTORY**

11466 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

-`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11467 **NAME**

11468 feof — test end-of-file indicator on a stream

11469 **SYNOPSIS**

11470 `#include <stdio.h>`

11471 `int feof(FILE *stream);`

11472 **DESCRIPTION**

11473 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11474 conflict between the requirements described here and the ISO C standard is unintentional. This
11475 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11476 The *feof*( ) function shall test the end-of-file indicator for the stream pointed to by *stream*.

11477 **RETURN VALUE**

11478 The *feof*( ) function shall return non-zero if and only if the end-of-file indicator is set for *stream*.

11479 **ERRORS**

11480 No errors are defined.

11481 **EXAMPLES**

11482 None.

11483 **APPLICATION USAGE**

11484 None.

11485 **RATIONALE**

11486 None.

11487 **FUTURE DIRECTIONS**

11488 None.

11489 **SEE ALSO**

11490 *clearerr*( ), *ferror*( ), *fopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

11491 **CHANGE HISTORY**

11492 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

feraiseexcept — raise floating-point exception

SYNOPSIS

```
#include <fenv.h>

int feraiseexcept(int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *feraiseexcept*() function shall attempt to raise the supported floating-point exceptions represented by the argument *excepts*. The order in which these floating-point exceptions are raised is unspecified. Whether the *feraiseexcept*() function additionally raises the inexact floating-point exception whenever it raises the overflow or underflow floating-point exception is implementation-defined.

RETURN VALUE

If the argument is zero or if all the specified exceptions were successfully raised, *feraiseexcept*() shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The effect is intended to be similar to that of floating-point exceptions raised by arithmetic operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

RATIONALE

Raising overflow or underflow is allowed to also raise inexact because on some architectures the only practical way to raise an exception is to execute an instruction that has the exception as a side effect. The function is not restricted to accept only valid coincident expressions for atomic operations, so the function can be used to raise exceptions accrued over several operations.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fegetexceptflag*(), *fesetexceptflag*(), *fetestexcept*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<fenv.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11530 **NAME**

11531 ferror — test error indicator on a stream

11532 **SYNOPSIS**

11533 `#include <stdio.h>`

11534 `int ferror(FILE *stream);`

11535 **DESCRIPTION**

11536 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11537 conflict between the requirements described here and the ISO C standard is unintentional. This
11538 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11539 The *ferror*( ) function shall test the error indicator for the stream pointed to by *stream*.

11540 **RETURN VALUE**

11541 The *ferror*( ) function shall return non-zero if and only if the error indicator is set for *stream*.

11542 **ERRORS**

11543 No errors are defined.

11544 **EXAMPLES**

11545 None.

11546 **APPLICATION USAGE**

11547 None.

11548 **RATIONALE**

11549 None.

11550 **FUTURE DIRECTIONS**

11551 None.

11552 **SEE ALSO**

11553 *clearerr*( ), *feof*( ), *fopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

11554 **CHANGE HISTORY**

11555 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

11556 **NAME**

11557 fesetenv — set current floating-point environment

11558 **SYNOPSIS**

11559 `#include <fenv.h>`

11560 `int fesetenv(const fenv_t *`*`envp`*`);`

11561 **DESCRIPTION**

11562 Refer to *fegetenv*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11563 **NAME**

11564 fesetexceptflag — set floating-point status flags

11565 **SYNOPSIS**

11566 `#include <fenv.h>`

11567 `int fesetexceptflag(const fexcept_t *`*`flagp`*`, int `*`excepts`*`);`

11568 **DESCRIPTION**

11569 Refer to *fegetexceptflag*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

11570 **NAME**

11571 fesetround — set current rounding direction

11572 **SYNOPSIS**

11573
```
#include <fenv.h>
```

11574
```
int fesetround(int round);
```

11575 **DESCRIPTION**

11576 Refer to *fegetround*( ).

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

fetestexcept — test floating-point exception flags

SYNOPSIS

```
#include <fenv.h>

int fetestexcept(int excepts);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fetestexcept*( ) function shall determine which of a specified subset of the floating-point exception flags are currently set. The *excepts* argument specifies the floating-point status flags to be queried.

RETURN VALUE

The *fetestexcept*( ) function shall return the value of the bitwise-inclusive OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in *excepts*.

ERRORS

No errors are defined.

EXAMPLES

The following example calls function *f*( ) if an invalid exception is set, and then function *g*( ) if an overflow exception is set:

```
#include <fenv.h>
/* ... */
{
    #pragma STDC FENV_ACCESS ON
    int set_excepts;
    feclearexcept(FE_INVALID | FE_OVERFLOW);
    // maybe raise exceptions
    set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
    if (set_excepts & FE_INVALID) f();
    if (set_excepts & FE_OVERFLOW) g();
    /* ... */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fegetexceptflag*( ), *feraiseexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fenv.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11619 **CHANGE HISTORY**

11620 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11621 **NAME**

11622 feupdateenv — update floating-point environment

11623 **SYNOPSIS**

11624
```
#include <fenv.h>
```

11625
```
int feupdateenv(const fenv_t *envp);
```

11626 **DESCRIPTION**

11627 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11628 conflict between the requirements described here and the ISO C standard is unintentional. This
11629 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11630 The *feupdateenv*( ) function shall attempt to save the currently raised floating-point exceptions in
11631 its automatic storage, attempt to install the floating-point environment represented by the object
11632 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument
11633 *envp* shall point to an object set by a call to *feholdexcept*( ) or *fegetenv*( ), or equal a floating-point
11634 environment macro.

11635 **RETURN VALUE**

11636 The *feupdateenv*( ) function shall return a zero value if and only if all the required actions were
11637 successfully carried out.

11638 **ERRORS**

11639 No errors are defined.

11640 **EXAMPLES**

11641 The following example shows sample code to hide spurious underflow floating-point
11642 exceptions:

11643
```
#include <fenv.h>
double f(double x)
{
    #pragma STDC FENV_ACCESS ON
    double result;
    fenv_t save_env;
    feholdexcept(&save_env);
    // compute result
    if (/* test spurious underflow */)
    feclearexcept(FE_UNDERFLOW);
    feupdateenv(&save_env);
    return result;
}
```

11656 **APPLICATION USAGE**

11657 None.

11658 **RATIONALE**

11659 None.

11660 **FUTURE DIRECTIONS**

11661 None.

11662 **SEE ALSO**

11663 *fegetenv*( ), *feholdexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fenv.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

11664 **CHANGE HISTORY**

11665 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

11666 ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11667 **NAME**

11668 fflush — flush a stream

11669 **SYNOPSIS**

11670 `#include <stdio.h>`

11671 `int fflush(FILE *stream);`

11672 **DESCRIPTION**

11673 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11674 conflict between the requirements described here and the ISO C standard is unintentional. This
11675 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11676 If *stream* points to an output stream or an update stream in which the most recent operation was
11677 CX not input, *fflush*( ) shall cause any unwritten data for that stream to be written to the file, and the
11678 *st_ctime* and *st_mtime* fields of the underlying file shall be marked for update.

11679 If *stream* is a null pointer, *fflush*( ) shall perform this flushing action on all streams for which the
11680 behavior is defined above.

11681 **RETURN VALUE**

11682 Upon successful completion, *fflush*( ) shall return 0; otherwise, it shall set the error indicator for
11683 CX the stream, return EOF, and set *errno* to indicate the error.

11684 **ERRORS**

11685 The *fflush*( ) function shall fail if:

11686 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
11687 process would be delayed in the write operation.

11688 CX [EBADF] The file descriptor underlying *stream* is not valid.

11689 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

11690 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

11691 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
11692 offset maximum associated with the corresponding stream.

11693 CX [EINTR] The *fflush*( ) function was interrupted by a signal.

11694 CX [EIO] The process is a member of a background process group attempting to write
11695 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
11696 blocking SIGTTOU, and the process group of the process is orphaned. This
11697 error may also be returned under implementation-defined conditions.

11698 CX [ENOSPC] There was no free space remaining on the device containing the file.

11699 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
11700 any process. A SIGPIPE signal shall also be sent to the thread.

11701 The *fflush*( ) function may fail if:

11702 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
11703 capabilities of the device.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Sending Prompts to Standard Output

The following example uses *printf*() calls to print a series of prompts for information the user must enter from standard input. The *fflush*() calls force the output to standard output. The *fflush*() function is used because standard output is usually buffered and the prompt may not immediately be printed on the output or terminal. The *gets*() calls read strings from standard input and place the results in variables, for use later in the program.

```
#include <stdio.h>
...
char user[100];
char oldpasswd[100];
char newpasswd[100];
...
printf("User name: ");
fflush(stdout);
gets(user);

printf("Old password: ");
fflush(stdout);
gets(oldpasswd);

printf("New password: ");
fflush(stdout);
gets(newpasswd);
...
```

APPLICATION USAGE

None.

RATIONALE

Data buffered by the system may make determining the validity of the position of the current file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush*() on streams open for *read*() is not mandated by IEEE Std 1003.1-2001.

FUTURE DIRECTIONS

None.

SEE ALSO

getrlimit(), *ulimit*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] error is added as part of the large file support extensions.
- The [ENXIO] optional error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11747 The RETURN VALUE section is updated to note that the error indicator shall be set for the
11748 stream. This is for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11749 **NAME**

11750 ffs — find first set bit

11751 **SYNOPSIS**

11752 XSI
```
#include <strings.h>
```

11753
```
int ffs(int i);
```

11754

11755 **DESCRIPTION**

11756 The *ffs*( ) function shall find the first bit set (beginning with the least significant bit) in *i*, and
11757 return the index of that bit. Bits are numbered starting at one (the least significant bit).

11758 **RETURN VALUE**

11759 The *ffs*( ) function shall return the index of the first bit set. If *i* is 0, then *ffs*( ) shall return 0.

11760 **ERRORS**

11761 No errors are defined.

11762 **EXAMPLES**

11763 None.

11764 **APPLICATION USAGE**

11765 None.

11766 **RATIONALE**

11767 None.

11768 **FUTURE DIRECTIONS**

11769 None.

11770 **SEE ALSO**

11771 The Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

11772 **CHANGE HISTORY**

11773 First released in Issue 4, Version 2.

11774 **Issue 5**

11775 Moved from X/OPEN UNIX extension to BASE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

fgetc — get a byte from a stream

SYNOPSIS

```
#include <stdio.h>

int fgetc(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is present, the *fgetc*( ) function shall obtain the next byte as an **unsigned char** converted to an **int**, from the input stream pointed to by *stream*, and advance the associated file position indicator for the stream (if defined). Since *fgetc*( ) operates on bytes, reading a character consisting of multiple bytes (or ''a multi-byte character'') may require multiple calls to *fgetc*( ).

CX The *fgetc*( ) function may mark the *st_atime* field of the file associated with *stream* for update. The *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fgetwc*( ), *fgetws*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *gets*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ) or *ungetwc*( ).

RETURN VALUE

Upon successful completion, *fgetc*( ) shall return the next byte from the input stream pointed to by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetc*( ) shall return EOF. If a read error occurs, CX the error indicator for the stream shall be set, *fgetc*( ) shall return EOF, and shall set *errno* to indicate the error.

ERRORS

The *fgetc*( ) function shall fail if data needs to be read and:

CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the process would be delayed in the *fgetc*( ) operation.

CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for reading.

CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

CX [EIO] A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group is orphaned. This error may also be generated for implementation-defined reasons.

CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

The *fgetc*( ) function may fail if:

CX [ENOMEM] Insufficient storage space is available.

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *fgetc*() is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

The *ferror*() or *feof*() functions must be used to distinguish between an error condition and an end-of-file condition.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feof(), *ferror*(), *fopen*(), *getchar*(), *getc*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the input stream is not set.
- The RETURN VALUE section is updated to note that the error indicator shall be set for the stream.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11848 **NAME**

11849 fgetpos — get current file position information

11850 **SYNOPSIS**

11851 `#include <stdio.h>`

11852 `int fgetpos(FILE *restrict stream, fpos_t *restrict pos);`

11853 **DESCRIPTION**

11854 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11855 conflict between the requirements described here and the ISO C standard is unintentional. This
11856 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11857 The *fgetpos*( ) function shall store the current values of the parse state (if any) and file position
11858 indicator for the stream pointed to by *stream* in the object pointed to by *pos*. The value stored
11859 contains unspecified information usable by *fsetpos*( ) for repositioning the stream to its position
11860 at the time of the call to *fgetpos*( ).

11861 **RETURN VALUE**

11862 Upon successful completion, *fgetpos*( ) shall return 0; otherwise, it shall return a non-zero value
11863 and set *errno* to indicate the error.

11864 **ERRORS**

11865 The *fgetpos*( ) function shall fail if:

11866 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an
11867 object of type **fpos_t**.

11868 The *fgetpos*( ) function may fail if:

11869 CX [EBADF] The file descriptor underlying *stream* is not valid.

11870 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

11871

11872 **EXAMPLES**

11873 None.

11874 **APPLICATION USAGE**

11875 None.

11876 **RATIONALE**

11877 None.

11878 **FUTURE DIRECTIONS**

11879 None.

11880 **SEE ALSO**

11881 *fopen*( ), *ftell*( ), *rewind*( ), *ungetc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

11882 **CHANGE HISTORY**

11883 First released in Issue 4. Derived from the ISO C standard.

11884 **Issue 5**

11885 Large File Summit extensions are added.

11886 **Issue 6**

11887 Extensions beyond the ISO C standard are marked.

11888 The following new requirements on POSIX implementations derive from alignment with the
11889 Single UNIX Specification:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The [EBADF] and [ESPIPE] optional error conditions are added.

An additional [ESPIPE] error condition is added for sockets.

The prototype for *fgetpos*( ) is changed for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

fgets — get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *fgets(char *restrict s, int n, FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fgets*( ) function shall read bytes from *stream* into the array pointed to by *s*, until *n*−1 bytes are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered. The string is then terminated with a null byte.

CX The *fgets*( ) function may mark the *st_atime* field of the file associated with *stream* for update. The *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fgetwc*( ), *fgetws*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *gets*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ) or *ungetwc*( ).

RETURN VALUE

Upon successful completion, *fgets*( ) shall return *s*. If the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgets*( ) shall return a null pointer. If a read error occurs, CX the error indicator for the stream shall be set, *fgets*( ) shall return a null pointer, and shall set *errno* to indicate the error.

ERRORS

Refer to *fgetc*( ).

EXAMPLES

Reading Input

The following example uses *fgets*( ) to read each line of input. {LINE_MAX}, which defines the maximum size of the input line, is defined in the **<limits.h>** header.

```
#include <stdio.h>
...
char line[LINE_MAX];
...
while (fgets(line, LINE_MAX, fp) != NULL) {
...
}
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11934 **SEE ALSO**

11935 *fopen*( ), *fread*( ), *gets*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

11936 **CHANGE HISTORY**

11937 First released in Issue 1. Derived from Issue 1 of the SVID.

11938 **Issue 6**

11939 Extensions beyond the ISO C standard are marked.

11940 The prototype for *fgets*( ) is changed for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11941 **NAME**

11942 fgetwc — get a wide-character code from a stream

11943 **SYNOPSIS**

11944 ```
#include <stdio.h>
```

11945 ```
#include <wchar.h>
```

11946 ```
wint_t fgetwc(FILE *stream);
```

11947 **DESCRIPTION**

11948 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11949 conflict between the requirements described here and the ISO C standard is unintentional. This
11950 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11951 The *fgetwc*( ) function shall obtain the next character (if present) from the input stream pointed to
11952 by *stream*, convert that to the corresponding wide-character code, and advance the associated
11953 file position indicator for the stream (if defined).

11954 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

11955 CX The *fgetwc*( ) function may mark the *st_atime* field of the file associated with *stream* for update.
11956 The *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ),
11957 *fgetwc*( ), *fgetws*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *gets*( ), or *scanf*( ) using *stream* that returns
11958 data not supplied by a prior call to *ungetc*( ) or *ungetwc*( ).

11959 **RETURN VALUE**

11960 Upon successful completion, the *fgetwc*( ) function shall return the wide-character code of the
11961 character read from the input stream pointed to by *stream* converted to a type **wint_t**. If the
11962 stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetwc*( ) shall
11963 return WEOF. If a read error occurs, the error indicator for the stream shall be set, *fgetwc*( ) shall
11964 CX return WEOF, and shall set *errno* to indicate the error. If an encoding error occurs, the error
11965 indicator for the stream shall be set, *fgetwc*( ) shall return WEOF, and shall set *errno* to indicate
11966 the error.

11967 **ERRORS**

11968 The *fgetwc*( ) function shall fail if data needs to be read and:

11969 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
11970 process would be delayed in the *fgetwc*( ) operation.

11971 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
11972 reading.

11973 [EILSEQ] The data obtained from the input stream does not form a valid character.

11974 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
11975 was transferred.

11976 CX [EIO] A physical I/O error has occurred, or the process is in a background process
11977 group attempting to read from its controlling terminal, and either the process
11978 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
11979 This error may also be generated for implementation-defined reasons.

11980 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
11981 offset maximum associated with the corresponding stream.

11982 The *fgetwc*( ) function may fail if:

11983 CX [ENOMEM] Insufficient storage space is available.


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
```

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

The *ferror*( ) or *feof*( ) functions must be used to distinguish between an error condition and an end-of-file condition.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feof( ), *ferror*( ), *fopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The Optional Header (OH) marking is removed from **<stdio.h>**.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12009 **NAME**

12010 fgetws — get a wide-character string from a stream

12011 **SYNOPSIS**

12012 `#include <stdio.h>`
12013 `#include <wchar.h>`

12014 `wchar_t *fgetws(wchar_t *restrict ws, int n,`
12015 `FILE *restrict stream);`

12016 **DESCRIPTION**

12017 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12018 conflict between the requirements described here and the ISO C standard is unintentional. This
12019 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12020 The *fgetws*( ) function shall read characters from the *stream*, convert these to the corresponding
12021 wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*−1 characters are
12022 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is
12023 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character
12024 code.

12025 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

12026 CX The *fgetws*( ) function may mark the *st_atime* field of the file associated with *stream* for update.
12027 The *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ),
12028 *fgetwc*( ), *fgetws*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *gets*( ), or *scanf*( ) using *stream* that returns
12029 data not supplied by a prior call to *ungetc*( ) or *ungetwc*( ).

12030 **RETURN VALUE**

12031 Upon successful completion, *fgetws*( ) shall return *ws*. If the stream is at end-of-file, the end-of-
12032 file indicator for the stream shall be set and *fgetws*( ) shall return a null pointer. If a read error
12033 CX occurs, the error indicator for the stream shall be set, *fgetws*( ) shall return a null pointer, and
12034 shall set *errno* to indicate the error.

12035 **ERRORS**

12036 Refer to *fgetwc*( ).

12037 **EXAMPLES**

12038 None.

12039 **APPLICATION USAGE**

12040 None.

12041 **RATIONALE**

12042 None.

12043 **FUTURE DIRECTIONS**

12044 None.

12045 **SEE ALSO**

12046 *fopen*( ), *fread*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<wchar.h>**

12047 **CHANGE HISTORY**

12048 First released in Issue 4. Derived from the MSE working draft.

12049 **Issue 5**

12050 The Optional Header (OH) marking is removed from **<stdio.h>**.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12051 **Issue 6**

12052 Extensions beyond the ISO C standard are marked.

12053 The prototype for *fgetws*( ) is changed for alignment with the ISO / IEC 9899: 1999 standard.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12054 **NAME**

12055 fileno — map a stream pointer to a file descriptor

12056 **SYNOPSIS**

12057 CX
```
#include <stdio.h>
```

12058
```
int fileno(FILE *stream);
```

12059

12060 **DESCRIPTION**

12061 The *fileno*( ) function shall return the integer file descriptor associated with the stream pointed to
12062 by *stream*.

12063 **RETURN VALUE**

12064 Upon successful completion, *fileno*( ) shall return the integer value of the file descriptor
12065 associated with *stream*. Otherwise, the value –1 shall be returned and *errno* set to indicate the
12066 error.

12067 **ERRORS**

12068 The *fileno*( ) function may fail if:

12069 [EBADF] The *stream* argument is not a valid stream.

12070 **EXAMPLES**

12071 None.

12072 **APPLICATION USAGE**

12073 None.

12074 **RATIONALE**

12075 Without some specification of which file descriptors are associated with these streams, it is
12076 impossible for an application to set up the streams for another application it starts with *fork*( )
12077 and *exec*. In particular, it would not be possible to write a portable version of the *sh* command
12078 interpreter (although there may be other constraints that would prevent that portability).

12079 **FUTURE DIRECTIONS**

12080 None.

12081 **SEE ALSO**

12082 Section 2.5.1 (on page 35), *fdopen*( ), *fopen*( ), *stdin*, the Base Definitions volume of
12083 IEEE Std 1003.1-2001, **<stdio.h>**

12084 **CHANGE HISTORY**

12085 First released in Issue 1. Derived from Issue 1 of the SVID.

12086 **Issue 6**

12087 The following new requirements on POSIX implementations derive from alignment with the
12088 Single UNIX Specification:

12089 • The [EBADF] optional error condition is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

flockfile, ftrylockfile, funlockfile — stdio locking functions

SYNOPSIS

TSF

```
#include <stdio.h>

void flockfile(FILE *file);
int ftrylockfile(FILE *file);
void funlockfile(FILE *file);
```

DESCRIPTION

These functions shall provide for explicit application-level locking of stdio (**FILE ***) objects. These functions can be used by a thread to delineate a sequence of I/O statements that are executed as a unit.

The *flockfile*( ) function shall acquire for a thread ownership of a (**FILE ***) object.

The *ftrylockfile*( ) function shall acquire for a thread ownership of a (**FILE ***) object if the object is available; *ftrylockfile*( ) is a non-blocking version of *flockfile*( ).

The *funlockfile*( ) function shall relinquish the ownership granted to the thread. The behavior is undefined if a thread other than the current owner calls the *funlockfile*( ) function.

The functions shall behave as if there is a lock count associated with each (**FILE ***) object. This count is implicitly initialized to zero when the (**FILE ***) object is created. The (**FILE ***) object is unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE ***) object. When the *flockfile*( ) function is called, if the count is zero or if the count is positive and the caller owns the (**FILE ***) object, the count shall be incremented. Otherwise, the calling thread shall be suspended, waiting for the count to return to zero. Each call to *funlockfile*( ) shall decrement the count. This allows matching calls to *flockfile*( ) (or successful calls to *ftrylockfile*( )) and *funlockfile*( ) to be nested.

All functions that reference (**FILE ***) objects shall behave as if they use *flockfile*( ) and *funlockfile*( ) internally to obtain ownership of these (**FILE ***) objects.

RETURN VALUE

None for *flockfile*( ) and *funlockfile*( ).

The *ftrylockfile*( ) function shall return zero for success and non-zero to indicate that the lock cannot be acquired.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

RATIONALE

The *flockfile*( ) and *funlockfile*( ) functions provide an orthogonal mutual-exclusion lock for each **FILE**. The *ftrylockfile*( ) function provides a non-blocking attempt to acquire a file lock, analogous to *pthread_mutex_trylock*( ).

These locks behave as if they are the same as those used internally by *stdio* for thread-safety. This both provides thread-safety of these functions without requiring a second level of internal locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Application writers and implementors should be aware that there are potential deadlock problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested via {_IOLBF}) require that certain input operations sometimes cause the buffered contents of implementation-defined line-buffered output streams to be flushed. If two threads each hold the lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can typically be avoided by acquiring locks on input streams before locks on output streams if a thread would be acquiring both.

In summary, threads sharing *stdio* streams with other threads can use *flockfile*( ) and *funlockfile*( ) to cause sequences of I/O performed by a single thread to be kept bundled. The only case where the use of *flockfile*( ) and *funlockfile*( ) is required is to provide a scope protecting uses of the **_unlocked*( ) functions/macros. This moves the cost/performance tradeoff to the optimal point.

FUTURE DIRECTIONS

None.

SEE ALSO

getc_unlocked( ), *putc_unlocked*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

These functions are marked as part of the Thread-Safe Functions option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

12155 **NAME**

12156 floor, floorf, floorl — floor function

12157 **SYNOPSIS**

12158
```
#include <math.h>
```

12159
```
double floor(double x);
```
12160
```
float floorf(float x);
```
12161
```
long double floorl(long double x);
```

12162 **DESCRIPTION**

12163 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12164 conflict between the requirements described here and the ISO C standard is unintentional. This
12165 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12166 These functions shall compute the largest integral value not greater than *x*.

12167 An application wishing to check for error situations should set *errno* to zero and call
12168 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
12169 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
12170 zero, an error has occurred.

12171 **RETURN VALUE**

12172 Upon successful completion, these functions shall return the largest integral value not greater
12173 than *x*, expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the
12174 function.

12175 MX If *x* is NaN, a NaN shall be returned.

12176 If *x* is ±0 or ±Inf, *x* shall be returned.

12177 XSI If the correct value would cause overflow, a range error shall occur and *floor*(), *floorf*(), and
12178 *floorl*() shall return the value of the macro −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL,
12179 respectively.

12180 **ERRORS**

12181 These functions shall fail if:

12182 XSI

Range Error	The result would cause an overflow.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

12187 **EXAMPLES**

12188 None.

12189 **APPLICATION USAGE**

12190 The integral value returned by these functions might not be expressible as an **int** or **long**. The
12191 return value should be tested before assigning it to an integer type to avoid the undefined results
12192 of an integer overflow.

12193 The *floor*() function can only overflow when the floating-point representation has
12194 DBL_MANT_DIG > DBL_MAX_EXP.

12195 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
12196 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12197 **RATIONALE**

12198 None.

12199 **FUTURE DIRECTIONS**

12200 None.

12201 **SEE ALSO**

12202 *ceil*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
12203 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

12204 **CHANGE HISTORY**

12205 First released in Issue 1. Derived from Issue 1 of the SVID.

12206 **Issue 5**

12207 The DESCRIPTION is updated to indicate how an application should check for an error. This
12208 text was previously published in the APPLICATION USAGE section.

12209 **Issue 6**

12210 The *floorf*( ) and *floorl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

12211 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
12212 revised to align with the ISO / IEC 9899: 1999 standard.

12213 IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are
12214 marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fma, fmaf, fmal — floating-point multiply-add

SYNOPSIS

```
#include <math.h>

double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
long double fmal(long double x, long double y, long double z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of FLT_ROUNDS.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary operation.

MX If x or y are NaN, a NaN shall be returned.

If x multiplied by y is an exact infinity and z is also an infinity but with the opposite sign, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If one of x and y is infinite, the other is zero, and z is not a NaN, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If one of x and y is infinite, the other is zero, and z is a NaN, a NaN shall be returned and a domain error may occur.

If $x*y$ is not 0*Inf nor Inf*0 and z is a NaN, a NaN shall be returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The value of $x*y+z$ is invalid, or the value $x*y$ is invalid and z is not a NaN.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.
Range Error	The result overflows.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

These functions may fail if:

MX

Domain Error	The value *x***y* is invalid and *z* is a NaN.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

MX

Range Error	The result underflows.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its unexpected use by the compiler can undermine carefully written code. The FP_CONTRACT macro can be used to disallow use of floating multiply-add; and the *fma*( ) function guarantees its use where desired. Many current machines provide hardware floating multiply-add instructions; software implementation can be used for others.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12286 **NAME**

12287 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

12288 **SYNOPSIS**

12289
```
#include <math.h>

double fmax(double x, double y);
float fmaxf(float x, float y);
long double fmaxl(long double x, long double y);
```

12293 **DESCRIPTION**

12294 CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12297 These functions shall determine the maximum numeric value of their arguments. NaN arguments shall be treated as missing data: if one argument is a NaN and the other numeric, then these functions shall choose the numeric value.

12300 **RETURN VALUE**

12301 Upon successful completion, these functions shall return the maximum numeric value of their arguments.

12303 If just one argument is a NaN, the other argument shall be returned.

12304 MX If *x* and *y* are NaN, a NaN shall be returned.

12305 **ERRORS**

12306 No errors are defined.

12307 **EXAMPLES**

12308 None.

12309 **APPLICATION USAGE**

12310 None.

12311 **RATIONALE**

12312 None.

12313 **FUTURE DIRECTIONS**

12314 None.

12315 **SEE ALSO**

12316 *fdim*( ), *fmin*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

12317 **CHANGE HISTORY**

12318 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12319 **NAME**

12320 fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

12321 **SYNOPSIS**

12322 ```
#include <math.h>
```

12323 ```
double fmin(double x, double y);
```
12324 ```
float fminf(float x, float y);
```
12325 ```
long double fminl(long double x, long double y);
```

12326 **DESCRIPTION**

12327 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12328 conflict between the requirements described here and the ISO C standard is unintentional. This
12329 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12330 These functions shall determine the minimum numeric value of their arguments. NaN
12331 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
12332 then these functions shall choose the numeric value.

12333 **RETURN VALUE**

12334 Upon successful completion, these functions shall return the minimum numeric value of their
12335 arguments.

12336 If just one argument is a NaN, the other argument shall be returned.

12337 MX If *x* and *y* are NaN, a NaN shall be returned.

12338 **ERRORS**

12339 No errors are defined.

12340 **EXAMPLES**

12341 None.

12342 **APPLICATION USAGE**

12343 None.

12344 **RATIONALE**

12345 None.

12346 **FUTURE DIRECTIONS**

12347 None.

12348 **SEE ALSO**

12349 *fdim*( ), *fmax*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

12350 **CHANGE HISTORY**

12351 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fmod, fmodf, fmodl — floating-point remainder value function

SYNOPSIS

```
#include <math.h>

double fmod(double x, double y);
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall return the floating-point remainder of the division of *x* by *y*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

These functions shall return the value $x-i*y$, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.

If the correct value would cause underflow, and is not representable, a range error may occur, and MX either 0.0 (if supported), or an implementation-defined value shall be returned.

MX If *x* or *y* is NaN, a NaN shall be returned.

If *y* is zero, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is infinite, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is ±0 and *y* is not zero, ±0 shall be returned.

If *x* is not infinite and *y* is ±Inf, *x* shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is infinite or *y* is zero.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Range Error	The result underflows.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12395 **EXAMPLES**

12396 None.

12397 **APPLICATION USAGE**

12398 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
12399 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

12400 **RATIONALE**

12401 None.

12402 **FUTURE DIRECTIONS**

12403 None.

12404 **SEE ALSO**

12405 *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
12406 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

12407 **CHANGE HISTORY**

12408 First released in Issue 1. Derived from Issue 1 of the SVID.

12409 **Issue 5**

12410 The DESCRIPTION is updated to indicate how an application should check for an error. This
12411 text was previously published in the APPLICATION USAGE section.

12412 **Issue 6**

12413 The behavior for when the *y* argument is zero is now defined.

12414 The *fmodf*( ) and *fmodl*( ) functions are added for alignment with the ISO / IEC 9899: 1999
12415 standard.

12416 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
12417 revised to align with the ISO / IEC 9899: 1999 standard.

12418 IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are
12419 marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fmtmsg — display a message in the specified format on standard error and/or a system console

SYNOPSIS

XSI
```
#include <fmtmsg.h>

int fmtmsg(long classification, const char *label, int severity,
    const char *text, const char *action, const char *tag);
```

DESCRIPTION

The *fmtmsg*( ) function shall display messages in a specified format instead of the traditional *printf*( ) function.

Based on a message's classification component, *fmtmsg*( ) shall write a formatted message either to standard error, to the console, or to both.

A formatted message consists of up to five components as defined below. The component *classification* is not part of a message displayed to the user, but defines the source of the message and directs the display of the formatted message.

classification Contains the sum of identifying values constructed from the constants defined below. Any one identifier from a subclass may be used in combination with a single identifier from a different subclass. Two or more identifiers from the same subclass should not be used together, with the exception of identifiers from the display subclass. (Both display subclass identifiers may be used so that messages can be displayed to both standard error and the system console.)

Major Classifications

Identifies the source of the condition. Identifiers are: MM_HARD (hardware), MM_SOFT (software), and MM_FIRM (firmware).

Message Source Subclassifications

Identifies the type of software in which the problem is detected. Identifiers are: MM_APPL (application), MM_UTIL (utility), and MM_OPSYS (operating system).

Display Subclassifications

Indicates where the message is to be displayed. Identifiers are: MM_PRINT to display the message on the standard error stream, MM_CONSOLE to display the message on the system console. One or both identifiers may be used.

Status Subclassifications

Indicates whether the application can recover from the condition. Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-recoverable).

An additional identifier, MM_NULLMC, indicates that no classification component is supplied for the message.

label Identifies the source of the message. The format is two fields separated by a colon. The first field is up to 10 bytes, the second is up to 14 bytes.

severity Indicates the seriousness of the condition. Identifiers for the levels of *severity* are:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

	MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string `"HALT"`.
	MM_ERROR	Indicates that the application has detected a fault. Produces the string `"ERROR"`.
	MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string `"WARNING"`.
	MM_INFO	Provides information about a condition that is not in error. Produces the string `"INFO"`.
	MM_NOSEV	Indicates that no severity level is supplied for the message.
text		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
action		Describes the first step to be taken in the error-recovery process. The *fmtmsg*( ) function precedes the action string with the prefix: `"TO FIX:"`. The *action* string is not limited to a specific size.
tag		An identifier that references on-line documentation for the message. Suggested usage is that *tag* includes the *label* and a unique identifying number. A sample *tag* is `"XSI:cat:146"`.

The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg*( ) which message components it is to select when writing messages to standard error. The value of *MSGVERB* shall be a colon-separated list of optional keywords. Valid keywords are: *label*, *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the component's value is not the component's null value, *fmtmsg*( ) shall include that component in the message when writing the message to standard error. If *MSGVERB* does not include a keyword for a message component, that component shall not be included in the display of the message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, *fmtmsg*( ) shall select all components.

MSGVERB shall determine which components are selected for display to standard error. All message components shall be included in console messages.

RETURN VALUE

The *fmtmsg*( ) function shall return one of the following values:

MM_OK	The function succeeded.
MM_NOTOK	The function failed completely.
MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.
MM_NOCON	The function was unable to generate a console message, but otherwise succeeded.

ERRORS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

1. The following example of *fmtmsg*( ):

```
fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",
"refer to cat in user's reference manual", "XSI:cat:001")
```

produces a complete message in the specified message format:

```
XSI:cat: ERROR: illegal option
TO FIX: refer to cat in user's reference manual XSI:cat:001
```

2. When the environment variable *MSGVERB* is set as follows:

```
MSGVERB=severity:text:action
```

and Example 1 is used, *fmtmsg*( ) produces:

```
ERROR: illegal option
TO FIX: refer to cat in user's reference manual
```

APPLICATION USAGE

One or more message components may be systematically omitted from messages generated by an application by using the null value of the argument for that component.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

printf( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fmtmsg.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`--`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12530 **NAME**

12531 fnmatch — match a filename or a pathname

12532 **SYNOPSIS**

12533 `#include <fnmatch.h>`

12534 `int fnmatch(const char *pattern, const char *string, int flags);`

12535 **DESCRIPTION**

12536 The *fnmatch*( ) function shall match patterns as described in the Shell and Utilities volume of
12537 IEEE Std 1003.1-2001, Section 2.13.1, Patterns Matching a Single Character, and Section 2.13.2,
12538 Patterns Matching Multiple Characters. It checks the string specified by the *string* argument to
12539 see if it matches the pattern specified by the *pattern* argument.

12540 The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-inclusive
12541 OR of zero or more of the flags defined in **<fnmatch.h>**. If the FNM_PATHNAME flag is set in
12542 *flags*, then a slash character (`'/'`) in *string* shall be explicitly matched by a slash in *pattern*; it shall
12543 not be matched by either the asterisk or question-mark special characters, nor by a bracket
12544 expression. If the FNM_PATHNAME flag is not set, the slash character shall be treated as an
12545 ordinary character.

12546 If FNM_NOESCAPE is not set in *flags*, a backslash character (`'\'`) in *pattern* followed by any
12547 other character shall match that second character in *string*. In particular, `"\\"` shall match a
12548 backslash in *string*. If FNM_NOESCAPE is set, a backslash character shall be treated as an
12549 ordinary character.

12550 If FNM_PERIOD is set in *flags*, then a leading period (`'.'`) in *string* shall match a period in
12551 *pattern*; as described by rule 2 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
12552 2.13.3, Patterns Used for Filename Expansion where the location of ‘‘leading’’ is indicated by the
12553 value of FNM_PATHNAME:

12554 • If FNM_PATHNAME is set, a period is ‘‘leading’’ if it is the first character in *string* or if it
12555 immediately follows a slash.

12556 • If FNM_PATHNAME is not set, a period is ‘‘leading’’ only if it is the first character of *string*.

12557 If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

12558 **RETURN VALUE**

12559 If *string* matches the pattern specified by *pattern*, then *fnmatch*( ) shall return 0. If there is no
12560 match, *fnmatch*( ) shall return FNM_NOMATCH, which is defined in **<fnmatch.h>**. If an error
12561 occurs, *fnmatch*( ) shall return another non-zero value.

12562 **ERRORS**

12563 No errors are defined.

12564 **EXAMPLES**

12565 None.

12566 **APPLICATION USAGE**

12567 The *fnmatch*( ) function has two major uses. It could be used by an application or utility that
12568 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of
12569 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that
12570 need to match strings in a similar manner.

12571 The name *fnmatch*( ) is intended to imply *filename* match, rather than *pathname* match. The default
12572 action of this function is to match filenames, rather than pathnames, since it gives no special
12573 significance to the slash character. With the FNM_PATHNAME flag, *fnmatch*( ) does match
12574 pathnames, but without tilde expansion, parameter expansion, or special treatment for a period

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

at the beginning of a filename.

RATIONALE

This function replaced the REG_FILENAME flag of *regcomp*( ) in early proposals of this volume of IEEE Std 1003.1-2001. It provides virtually the same functionality as the *regcomp*( ) and *regexec*( ) functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag was proposed for *regcomp*( ), and would have had the opposite effect from FNM_PATHNAME), but with a simpler function and less system overhead.

FUTURE DIRECTIONS

None.

SEE ALSO

glob( ), *wordexp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fnmatch.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12591 **NAME**

12592 fopen — open a stream

12593 **SYNOPSIS**

12594 `#include <stdio.h>`

12595 `FILE *fopen(const char *restrict filename, const char *restrict mode);`

12596 **DESCRIPTION**

12597 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12598 conflict between the requirements described here and the ISO C standard is unintentional. This
12599 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12600 The *fopen*( ) function shall open the file whose pathname is the string pointed to by *filename*, and
12601 associates a stream with it.

12602 The *mode* argument points to a string. If the string is one of the following, the file shall be opened
12603 in the indicated mode. Otherwise, the behavior is undefined.

12604 *r* or *rb*	Open file for reading.
12605 *w* or *wb*	Truncate to zero length or create file for writing.
12606 *a* or *ab*	Append; open or create file for writing at end-of-file.
12607 *r+* or *rb+* or *r+b*	Open file for update (reading and writing).
12608 *w+* or *wb+* or *w+b*	Truncate to zero length or create file for update.
12609 *a+* or *ab+* or *a+b*	Append; open or create file for update, writing at end-of-file.

12610 CX The character `'b'` shall have no effect, but is allowed for ISO C standard conformance. Opening
12611 a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not
12612 exist or cannot be read.

12613 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all
12614 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
12615 calls to *fseek*( ).

12616 When a file is opened with update mode (`'+'` as the second or third character in the *mode*
12617 argument), both input and output may be performed on the associated stream. However, the
12618 application shall ensure that output is not directly followed by input without an intervening call
12619 to *fflush*( ) or to a file positioning function (*fseek*( ), *fsetpos*( ), or *rewind*( )), and input is not directly
12620 followed by output without an intervening call to a file positioning function, unless the input
12621 operation encounters end-of-file.

12622 When opened, a stream is fully buffered if and only if it can be determined not to refer to an
12623 interactive device. The error and end-of-file indicators for the stream shall be cleared.

12624 CX If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, upon
12625 successful completion, the *fopen*( ) function shall mark for update the *st_atime*, *st_ctime*, and
12626 *st_mtime* fields of the file and the *st_ctime* and *st_mtime* fields of the parent directory.

12627 If *mode* is *w*, *wb*, *w+*, *wb+*, or *w+b*, and the file did previously exist, upon successful completion,
12628 *fopen*( ) shall mark for update the *st_ctime* and *st_mtime* fields of the file. The *fopen*( ) function
12629 shall allocate a file descriptor as *open*( ) does.

12630 XSI After a successful call to the *fopen*( ) function, the orientation of the stream shall be cleared, the
12631 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
12632 initial conversion state.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12633 CX The largest value that can be represented correctly in an object of type **off_t** shall be established
12634 as the offset maximum in the open file description.

12635 **RETURN VALUE**

12636 Upon successful completion, *fopen*( ) shall return a pointer to the object controlling the stream.
12637 CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

12638 **ERRORS**

12639 The *fopen*( ) function shall fail if:

12640 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
12641 exists and the permissions specified by *mode* are denied, or the file does not
12642 exist and write permission is denied for the parent directory of the file to be
12643 created.

12644 CX [EINTR] A signal was caught during *fopen*( ).

12645 CX [EISDIR] The named file is a directory and *mode* requires write access.

12646 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
12647 argument.

12648 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

12649 CX [ENAMETOOLONG]
12650 The length of the *filename* argument exceeds {PATH_MAX} or a pathname
12651 component is longer than {NAME_MAX}.

12652 CX [ENFILE] The maximum allowable number of files is currently open in the system.

12653 CX [ENOENT] A component of *filename* does not name an existing file or *filename* is an empty
12654 string.

12655 CX [ENOSPC] The directory or file system that would contain the new file cannot be
12656 expanded, the file does not exist, and the file was to be created.

12657 CX [ENOTDIR] A component of the path prefix is not a directory.

12658 CX [ENXIO] The named file is a character special or block special file, and the device
12659 associated with this special file does not exist.

12660 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented
12661 correctly in an object of type **off_t**.

12662 CX [EROFS] The named file resides on a read-only file system and *mode* requires write
12663 access.

12664 The *fopen*( ) function may fail if:

12665 CX [EINVAL] The value of the *mode* argument is not valid.

12666 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
12667 resolution of the *path* argument.

12668 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

12669 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

12670 CX [ENAMETOOLONG]
12671 Pathname resolution of a symbolic link produced an intermediate result
12672 whose length exceeds {PATH_MAX}.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12673 CX [ENOMEM] Insufficient storage space is available.

12674 CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode*
12675 requires write access.

12676 **EXAMPLES**

12677 **Opening a File**

12678 The following example tries to open the file named **file** for reading. The *fopen*( ) function returns
12679 a file pointer that is used in subsequent *fgets*( ) and *fclose*( ) calls. If the program cannot open the
12680 file, it just ignores it.

```
12681 #include <stdio.h>
12682 ...
12683 FILE *fp;
12684 ...
12685 void rgrep(const char *file)
12686 {
12687 ...
12688     if ((fp = fopen(file, "r")) == NULL)
12689         return;
12690 ...
12691 }
```

12692 **APPLICATION USAGE**

12693 None.

12694 **RATIONALE**

12695 None.

12696 **FUTURE DIRECTIONS**

12697 None.

12698 **SEE ALSO**

12699 *fclose*( ), *fdopen*( ), *freopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

12700 **CHANGE HISTORY**

12701 First released in Issue 1. Derived from Issue 1 of the SVID.

12702 **Issue 5**

12703 Large File Summit extensions are added.

12704 **Issue 6**

12705 Extensions beyond the ISO C standard are marked.

12706 The following new requirements on POSIX implementations derive from alignment with the
12707 Single UNIX Specification:

12708 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
12709 description. This change is to support large files.

12710 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
12711 large files.

12712 • The [ELOOP] mandatory error condition is added.

12713 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error
12714 conditions are added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototype for *fopen*() is updated.
- The DESCRIPTION is updated to note that if the argument *mode* points to a string other than those listed, then the behavior is undefined.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12722 **NAME**

12723 fork — create a new process

12724 **SYNOPSIS**

12725
```
#include <unistd.h>
```

12726
```
pid_t fork(void);
```

12727 **DESCRIPTION**

12728 The *fork*( ) function shall create a new process. The new process (child process) shall be an exact
12729 copy of the calling process (parent process) except as detailed below:

12730 • The child process shall have a unique process ID.

12731 • The child process ID also shall not match any active process group ID.

12732 • The child process shall have a different parent process ID, which shall be the process ID of
12733 the calling process.

12734 • The child process shall have its own copy of the parent's file descriptors. Each of the child's
12735 file descriptors shall refer to the same open file description with the corresponding file
12736 descriptor of the parent.

12737 • The child process shall have its own copy of the parent's open directory streams. Each open
12738 directory stream in the child process may share directory stream positioning with the
12739 corresponding directory stream of the parent.

12740 XSI • The child process shall have its own copy of the parent's message catalog descriptors.

12741 • The child process' values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to 0.

12742 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be
12743 canceled; see *alarm*( ).

12744 XSI • All *semadj* values shall be cleared.

12745 • File locks set by the parent process shall not be inherited by the child process.

12746 • The set of signals pending for the child process shall be initialized to the empty set.

12747 XSI • Interval timers shall be reset in the child process.

12748 SEM • Any semaphores that are open in the parent process shall also be open in the child process.

12749 ML • The child process shall not inherit any address space memory locks established by the parent
12750 process via calls to *mlockall*( ) or *mlock*( ).

12751 MF|SHM • Memory mappings created in the parent shall be retained in the child process.
12752 MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE mappings
12753 in the child, and any modifications to the data in these mappings made by the parent prior to
12754 calling *fork*( ) shall be visible to the child. Any modifications to the data in MAP_PRIVATE
12755 mappings made by the parent after *fork*( ) returns shall be visible only to the parent.
12756 Modifications to the data in MAP_PRIVATE mappings made by the child shall be visible only
12757 to the child.

12758 PS • For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the
12759 policy and priority settings of the parent process during a *fork*( ) function. For other
12760 scheduling policies, the policy and priority settings on *fork*( ) are implementation-defined.

12761 TMR • Per-process timers created by the parent shall not be inherited by the child process.

12762 MSG • The child process shall have its own copy of the message queue descriptors of the parent.
12763 Each of the message descriptors of the child shall refer to the same open message queue

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

description as the corresponding message descriptor of the parent.

AIO
- No asynchronous input or asynchronous output operations shall be inherited by the child process.

- A process shall be created with a single thread. If a multi-threaded process calls *fork*(), the new process shall contain a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources. Consequently, to avoid errors, the child process may only execute async-signal-safe operations until such time as one of the *exec* functions is called. Fork handlers may be established by means of the *pthread_atfork*() function in order to maintain application invariants across *fork*() calls.

  When the application calls *fork*() from a signal handler and any of the fork handlers registered by *pthread_atfork*() calls a function that is not asynch-signal-safe, the behavior is undefined.

TRC TRI
- If the Trace option and the Trace Inherit option are both supported:

  If the calling process was being traced in a trace stream that had its inheritance policy set to POSIX_TRACE_INHERITED, the child process shall be traced into that trace stream, and the child process shall inherit the parent's mapping of trace event names to trace event type identifiers. If the trace stream in which the calling process was being traced had its inheritance policy set to POSIX_TRACE_CLOSE_FOR_CHILD, the child process shall not be traced into that trace stream. The inheritance policy is set by a call to the *posix_trace_attr_setinherited*() function.

TRC
- If the Trace option is supported, but the Trace Inherit option is not supported:

  The child process shall not be traced into any of the trace streams of its parent process.

TRC
- If the Trace option is supported, the child process of a trace controller process shall not control the trace streams controlled by its parent process.

CPT
- The initial value of the CPU-time clock of the child process shall be set to zero.

TCT
- The initial value of the CPU-time clock of the single thread of the child process shall be set to zero.

All other process characteristics defined by IEEE Std 1003.1-2001 shall be the same in the parent and child processes. The inheritance of process characteristics not defined by IEEE Std 1003.1-2001 is unspecified by IEEE Std 1003.1-2001.

After *fork*(), both the parent and the child processes shall be capable of executing independently before either one terminates.

RETURN VALUE

Upon successful completion, *fork*() shall return 0 to the child process and shall return the process ID of the child process to the parent process. Both processes shall continue to execute from the *fork*() function. Otherwise, –1 shall be returned to the parent process, no child process shall be created, and *errno* shall be set to indicate the error.

ERRORS

The *fork*() function shall fail if:

[EAGAIN] The system lacked the necessary resources to create another process, or the system-imposed limit on the total number of processes under execution system-wide or by a single user {CHILD_MAX} would be exceeded.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *fork*( ) function may fail if:

[ENOMEM] Insufficient storage space is available.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Many historical implementations have timing windows where a signal sent to a process group (for example, an interactive SIGINT) just prior to or during execution of *fork*( ) is delivered to the parent following the *fork*( ) but not to the child because the *fork*( ) code clears the child's set of pending signals. This volume of IEEE Std 1003.1-2001 does not require, or even permit, this behavior. However, it is pragmatic to expect that problems of this nature may continue to exist in implementations that appear to conform to this volume of IEEE Std 1003.1-2001 and pass available verification suites. This behavior is only a consequence of the implementation failing to make the interval between signal generation and delivery totally invisible. From the application's perspective, a *fork*( ) call should appear atomic. A signal that is generated prior to the *fork*( ) should be delivered prior to the *fork*( ). A signal sent to the process group after the *fork*( ) should be delivered to both parent and child. The implementation may actually initialize internal data structures corresponding to the child's set of pending signals to include signals sent to the process group during the *fork*( ). Since the *fork*( ) call can be considered as atomic from the application's perspective, the set would be initialized as empty and such signals would have arrived after the *fork*( ); see also **<signal.h>**.

One approach that has been suggested to address the problem of signal inheritance across *fork*( ) is to add an [EINTR] error, which would be returned when a signal is detected during the call. While this is preferable to losing signals, it was not considered an optimal solution. Although it is not recommended for this purpose, such an error would be an allowable extension for an implementation.

The [ENOMEM] error value is reserved for those implementations that detect and distinguish such a condition. This condition occurs when an implementation detects that there is not enough memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate because there can never be enough memory (either primary or secondary storage) to perform the operation. Since *fork*( ) duplicates an existing process, this must be a condition where there is sufficient memory for one such process, but not for two. Many historical implementations actually return [ENOMEM] due to temporary lack of memory, a case that is not generally distinct from [EAGAIN] from the perspective of a conforming application.

Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it and it should be reserved for the error condition specified there. The condition is not applicable on many implementations.

IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork*( ). A system that single-threads processes was clearly not intended and is considered an unacceptable ''toy implementation'' of this volume of IEEE Std 1003.1-2001. The only objection anticipated to the phrase ''executing independently'' is testability, but this assertion should be testable. Such tests require that both the parent and child can block on a detectable action of the other, such as a write to a pipe or a signal. An interactive exchange of such actions should be possible for the system to conform to the intent of this volume of IEEE Std 1003.1-2001.

The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it occurs or not is not in any practical sense under the control of the application because the condition is usually a consequence of the user's use of the system, not of the application's code.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Thus, no application can or should rely upon its occurrence under any circumstances, nor should the exact semantics of what concept of ''user'' is used be of concern to the application writer. Validation writers should be cognizant of this limitation.

There are two reasons why POSIX programmers call *fork*(). One reason is to create a new thread of control within the same program (which was originally only possible in POSIX by creating a new process); the other is to create a new process running a different program. In the latter case, the call to *fork*() is soon followed by a call to one of the *exec* functions.

The general problem with making *fork*() work in a multi-threaded world is what to do with all of the threads. There are two alternatives. One is to copy all of the threads into the new process. This causes the programmer or implementation to deal with threads that are suspended on system calls or that might be about to execute system calls that should not be executed in the new process. The other alternative is to copy only the thread that calls *fork*(). This creates the difficulty that the state of process-local resources is usually held in process memory. If a thread that is not calling *fork*() holds a resource, that resource is never released in the child process because the thread whose job it is to release the resource does not exist in the child process.

When a programmer is writing a multi-threaded program, the first described use of *fork*(), creating new threads in the same program, is provided by the *pthread_create*() function. The *fork*() function is thus used only to run new programs, and the effects of calling functions that require certain resources between the call to *fork*() and the call to an *exec* function are undefined.

The addition of the *forkall*() function to the standard was considered and rejected. The *forkall*() function lets all the threads in the parent be duplicated in the child. This essentially duplicates the state of the parent in the child. This allows threads in the child to continue processing and allows locks and the state to be preserved without explicit *pthread_atfork*() code. The calling process has to ensure that the threads processing state that is shared between the parent and child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall*(). For example, if a thread is reading a file descriptor in the parent when *forkall*() is called, then two threads (one in the parent and one in the child) are reading the file descriptor after the *forkall*(). If this is not desired behavior, the parent process has to synchronize with such threads before calling *forkall*().

While the *fork*() function is async-signal-safe, there is no way for an implementation to determine whether the fork handlers established by *pthread_atfork*() are async-signal-safe. The fork handlers may attempt to execute portions of the implementation that are not async-signal-safe, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore undefined for the fork handlers to execute functions that are not async-signal-safe when *fork*() is called from a signal handler.

When *forkall*() is called, threads, other than the calling thread, that are in functions that can return with an [EINTR] error may have those functions return [EINTR] if the implementation cannot ensure that the function behaves correctly in the parent and child. In particular, *pthread_cond_wait*() and *pthread_cond_timedwait*() need to return in order to ensure that the condition has not changed. These functions can be awakened by a spurious condition wakeup rather than returning [EINTR].

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *exec*, *fcntl*(), *posix_trace_attr_getinherited*(), *posix_trace_trid_eventid_open*(), *pthread_atfork*(), *semop*(), *signal*(), *times*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of *fork*( ) on a pending alarm call in the child process is clarified.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the DESCRIPTION and RATIONALE relating to fork handlers registered by the *pthread_atfork*( ) function and async-signal safety.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fpathconf, pathconf — get configurable pathname variables

SYNOPSIS

```
#include <unistd.h>

long fpathconf(int fildes, int name);
long pathconf(const char *path, int name);
```

DESCRIPTION

The *fpathconf*( ) and *pathconf*( ) functions shall determine the current value of a configurable limit or option (*variable*) that is associated with a file or directory.

For *pathconf*( ), the *path* argument points to the pathname of a file or directory.

For *fpathconf*( ), the *fildes* argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to that file or directory. Implementations shall support all of the variables listed in the following table and may support others. The variables in the following table come from **<limits.h>** or **<unistd.h>** and the symbolic constants, defined in **<unistd.h>**, are the corresponding values used for *name*.

Variable	Value of *name*	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	3, 4
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3, 4
{PATH_MAX}	_PC_PATH_MAX	4, 5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4, 9
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3, 4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8

Requirements

1. If *path* or *fildes* refers to a directory, the value returned shall apply to the directory itself.
2. If *path* or *fildes* does not refer to a terminal file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
3. If *path* or *fildes* refers to a directory, the value returned shall apply to filenames within the directory.
4. If *path* or *fildes* does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of a relative pathname when the specified directory is the working directory.

6. If *path* refers to a FIFO, or *fildes* refers to a pipe or FIFO, the value returned shall apply to the referenced object. If *path* or *fildes* refers to a directory, the value returned shall apply to any FIFO that exists or can be created within the directory. If *path* or *fildes* refers to any other type of file, it is unspecified whether an implementation supports an association of the variable name with the specified file.

7. If *path* or *fildes* refers to a directory, the value returned shall apply to any files, other than directories, that exist or can be created within the directory.

8. If *path* or *fildes* refers to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.

9. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of the string that a symbolic link in that directory can contain.

RETURN VALUE

If *name* is an invalid value, both *pathconf*( ) and *fpathconf*( ) shall return –1 and set *errno* to indicate the error.

If the variable corresponding to *name* has no limit for the *path* or file descriptor, both *pathconf*( ) and *fpathconf*( ) shall return –1 without changing *errno*. If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf*( ) shall return –1 and set *errno* to indicate the error.

If the implementation needs to use *fildes* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid file descriptor, *fpathconf*( ) shall return –1 and set *errno* to indicate the error.

Otherwise, *pathconf*( ) or *fpathconf*( ) shall return the current variable value for the file or directory without changing *errno*. The value returned shall not be more restrictive than the corresponding value available to the application when it was compiled with the implementation's **<limits.h>** or **<unistd.h>**.

ERRORS

The *pathconf*( ) function shall fail if:

[EINVAL] The value of *name* is not valid.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

The *pathconf*( ) function may fail if:

[EACCES] Search permission is denied for a component of the path prefix.

[EINVAL] The implementation does not support an association of the variable *name* with the specified file.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

The *fpathconf*( ) function shall fail if:

[EINVAL] The value of *name* is not valid.

The *fpathconf*( ) function may fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] The implementation does not support an association of the variable *name* with the specified file.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pathconf*( ) function was proposed immediately after the *sysconf*( ) function when it was realized that some configurable values may differ across file system, directory, or device boundaries.

For example, {NAME_MAX} frequently changes between System V and BSD-based file systems; System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file systems, an application would be forced to limit all pathname components to 14 bytes, as this would be the value specified in **<limits.h>** on such a system.

Therefore, various useful values can be queried on any pathname or file descriptor, assuming that the appropriate permissions are in place.

The value returned for the variable {PATH_MAX} indicates the longest relative pathname that could be given if the specified directory is the process' current working directory. A process may not always be able to generate a name that long and use it if a subdirectory in the pathname crosses into a more restrictive file system.

The value returned for the variable _POSIX_CHOWN_RESTRICTED also applies to directories that do not have file systems mounted on them. The value may change when crossing a mount point, so applications that need to know should check for each directory. (An even easier check is to try the *chown*( ) function and look for an error in case it happens.)

Unlike the values returned by *sysconf*( ), the pathname-oriented variables are potentially more volatile and are not guaranteed to remain constant throughout the process' lifetime. For example, in between two calls to *pathconf*( ), the file system in question may have been unmounted and remounted with different characteristics.

Also note that most of the errors are optional. If one of the variables always has the same value on an implementation, the implementation need not look at *path* or *fildes* to return that value and is, therefore, not required to detect any of the errors except the meaning of [EINVAL] that indicates that the value of *name* is not valid for that variable.

If the value of any of the limits is unspecified (logically infinite), they will not be defined in **<limits.h>** and the *pathconf*( ) and *fpathconf*( ) functions return –1 without changing *errno*. This can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

to [EINVAL] in this case.

Since –1 is a valid return value for the *pathconf*( ) and *fpathconf*( ) functions, applications should set *errno* to zero before calling them and check *errno* only if the return value is –1.

For the case of {SYMLINK_MAX}, since both *pathconf*( ) and *open*( ) follow symbolic links, there is no way that *path* or *fildes* could refer to a symbolic link.

FUTURE DIRECTIONS

None.

SEE ALSO

confstr( ), *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<unistd.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to include {FILESIZEBITS}.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The _PC_SYMLINK_MAX entry is added to the table in the DESCRIPTION.

The following *pathconf*( ) variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth paragraph of the DESCRIPTION and removing shading and margin markers from the table. This change is needed since implementations are required to support all of these symbols.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13083 **NAME**

13084 fpclassify — classify real floating type

13085 **SYNOPSIS**

13086 `#include <math.h>`

13087 `int fpclassify(real-floating x);`

13088 **DESCRIPTION**

13089 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13090 conflict between the requirements described here and the ISO C standard is unintentional. This
13091 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13092 The *fpclassify*( ) macro shall classify its argument value as NaN, infinite, normal, subnormal,
13093 zero, or into another implementation-defined category. First, an argument represented in a
13094 format wider than its semantic type is converted to its semantic type. Then classification is based
13095 on the type of the argument.

13096 **RETURN VALUE**

13097 The *fpclassify*( ) macro shall return the value of the number classification macro appropriate to
13098 the value of its argument.

13099 **ERRORS**

13100 No errors are defined.

13101 **EXAMPLES**

13102 None.

13103 **APPLICATION USAGE**

13104 None.

13105 **RATIONALE**

13106 None.

13107 **FUTURE DIRECTIONS**

13108 None.

13109 **SEE ALSO**

13110 *isfinite*( ), *isinf*( ), *isnan*( ), *isnormal*( ), *signbit*( ), the Base Definitions volume of
13111 IEEE Std 1003.1-2001, **<math.h>**

13112 **CHANGE HISTORY**

13113 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

fprintf, printf, snprintf, sprintf — print formatted output

SYNOPSIS

```
#include <stdio.h>

int fprintf(FILE *restrict stream, const char *restrict format, ...);
int printf(const char *restrict format, ...);
int snprintf(char *restrict s, size_t n,
    const char *restrict format, ...);
int sprintf(char *restrict s, const char *restrict format, ...);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fprintf*( ) function shall place output on the named output *stream*. The *printf*( ) function shall place output on the standard output stream *stdout*. The *sprintf*( ) function shall place output followed by the null byte, `'\0'`, in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough space is available.

The *snprintf*( ) function shall be equivalent to *sprintf*( ), with the addition of the *n* argument which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s* may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of being written to the array, and a null byte is written at the end of the bytes actually written into the array.

If copying takes place between objects that overlap as a result of a call to *sprintf*( ) or *snprintf*( ), the results are undefined.

Each of these functions converts, formats, and prints its arguments under control of the *format*. The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is composed of zero or more directives: *ordinary characters*, which are simply copied to the output stream, and *conversion specifications*, each of which shall result in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are otherwise ignored.

XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character `%` (see below) is replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The *format* can contain either numbered argument conversion specifications (that is, `"%n$"` and `"*m$"`), or unnumbered argument conversion specifications (that is, `%` and `*`), but not both. The only exception to this is that `%%` can be mixed with the `"%n$"` form. The results of mixing numbered and unnumbered argument specifications in a *format* string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*–*1*)th, are specified in the format string.

In format strings containing the `"%n$"` form of conversion specification, numbered arguments in the argument list can be referenced from the format string as many times as required.

In format strings containing the `%` form of conversion specification, each conversion specification uses the first unused argument in the argument list.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CX All forms of the *fprintf*() functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period (`'.'`).

XSI Each conversion specification is introduced by the `'%'` character or by the character sequence `"%n$"`, after which the following appear in sequence:

- Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- An optional minimum *field width*. If the converted value has fewer bytes than the field width, it shall be padded with spaces by default on the left; it shall be padded on the right if the left-adjustment flag (`'-'`), described below, is given to the field width. The field width takes the form of an asterisk (`'*'`), described below, or a decimal integer.
- An optional *precision* that gives the minimum number of digits to appear for the `d`, `i`, `o`, `u`, `x`, and `X` conversion specifiers; the number of digits to appear after the radix character for the `a`, `A`, `e`, `E`, `f`, and `F` conversion specifiers; the maximum number of significant digits for the `g` and `G` conversion specifiers; or the maximum number of bytes to be printed from a string in the `s` and `S` conversion specifiers. The precision takes the form of a period (`'.'`) followed either by an asterisk (`'*'`), described below, or an optional decimal digit string, where a null digit string is treated as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional length modifier that specifies the size of the argument.
- A *conversion specifier* character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an asterisk (`'*'`). In this case an argument of type **int** supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a `'-'` flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the `"%n$"` form of a conversion specification, a field width or precision may be indicated by the sequence `"*m$"`, where *m* is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an integer argument containing the field width or precision, for example:

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The flag characters and their meanings are:

`'` The integer portion of the result of a decimal conversion (`%i`, `%d`, `%u`, `%f`, `%F`, `%g`, or `%G`) shall be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.

`-` The result of the conversion shall be left-justified within the field. The conversion is right-justified if this flag is not specified.

`+` The result of a signed conversion shall always begin with a sign (`'+'` or `'-'`). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.

<space> If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and `'+'` flags both appear, the <space> flag shall be ignored.

`#` Specifies that the value is to be converted to an alternative form. For `o` conversion, it increases the precision (if necessary) to force the first digit of the result to be zero. For `x`

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversion specifiers, trailing zeros shall *not* be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.

0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the '0' and '−' flags both appear, the '0' flag is ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag is ignored. XSI If the '0' and ''' flags both appear, the grouping characters are inserted before zero padding. For other conversions, the behavior is undefined.

The length modifiers and their meanings are:

hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char** or **unsigned char** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **signed char** or **unsigned char** before printing); or that a following n conversion specifier applies to a pointer to a **signed char** argument.

h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or **unsigned short** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **short** or **unsigned short** before printing); or that a following n conversion specifier applies to a pointer to a **short** argument.

l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or **unsigned long** argument; that a following n conversion specifier applies to a pointer to a **long** argument; that a following c conversion specifier applies to a **wint_t** argument; that a following s conversion specifier applies to a pointer to a **wchar_t** argument; or has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long** or **unsigned long long** argument; or that a following n conversion specifier applies to a pointer to a **long long** argument.

j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax_t** or **uintmax_t** argument; or that a following n conversion specifier applies to a pointer to an **intmax_t** argument.

z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size_t** or the corresponding signed integer type argument; or that a following n conversion specifier applies to a pointer to a signed integer type corresponding to a **size_t** argument.

t Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **ptrdiff_t** or the corresponding **unsigned** type argument; or that a following n conversion specifier applies to a pointer to a **ptrdiff_t** argument.

L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **long double** argument.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The conversion specifiers and their meanings are:

d, i — The **int** argument shall be converted to a signed decimal in the style "[−]*dddd*". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

o — The **unsigned** argument shall be converted to unsigned octal format in the style "*dddd*". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

u — The **unsigned** argument shall be converted to unsigned decimal format in the style "*dddd*". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

x — The **unsigned** argument shall be converted to unsigned hexadecimal format in the style "*dddd*"; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

X — Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".

f, F — The **double** argument shall be converted to decimal notation in the style "[−]*ddd.ddd*", where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit appears before it. The low-order digit shall be rounded in an implementation-defined manner.

A **double** argument representing an infinity shall be converted in one of the styles "[−]inf" or "[−]infinity"; which style is implementation-defined. A **double** argument representing a NaN shall be converted in one of the styles "[−]nan(*n-char-sequence*)" or "[−]nan"; which style, and the meaning of any *n-char-sequence*, is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.

e, E — The **double** argument shall be converted in the style "[−]*d.ddd*±*dd*", where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall appear. The low-order digit shall be rounded in an implementation-defined manner. The E conversion specifier shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.

g, G — The **double** argument shall be converted in the style f or e (or in the style F or E in the case of a G conversion specifier), with the precision specifying the number of significant

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13298 digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on
13299 the value converted; style `e` (or `E`) shall be used only if the exponent resulting from
13300 such a conversion is less than –4 or greater than or equal to the precision. Trailing zeros
13301 shall be removed from the fractional portion of the result; a radix character shall appear
13302 only if it is followed by a digit or a `'#'` flag is present.

13303 A **double** argument representing an infinity or NaN shall be converted in the style of
13304 an `f` or `F` conversion specifier.

13305 `a`, `A` A **double** argument representing a floating-point number shall be converted in the
13306 style `"[–]0x`*h*`.`*hhhh*`p±`*d*`"`, where there is one hexadecimal digit (which shall be non-
13307 zero if the argument is a normalized floating-point number and is otherwise
13308 unspecified) before the decimal-point character and the number of hexadecimal digits
13309 after it is equal to the precision; if the precision is missing and FLT_RADIX is a power
13310 of 2, then the precision shall be sufficient for an exact representation of the value; if the
13311 precision is missing and FLT_RADIX is not a power of 2, then the precision shall be
13312 sufficient to distinguish values of type **double**, except that trailing zeros may be
13313 omitted; if the precision is zero and the `'#'` flag is not specified, no decimal-point
13314 character shall appear. The letters `"abcdef"` shall be used for `a` conversion and the
13315 letters `"ABCDEF"` for `A` conversion. The `A` conversion specifier produces a number with
13316 `'X'` and `'P'` instead of `'x'` and `'p'`. The exponent shall always contain at least one
13317 digit, and only as many more digits as necessary to represent the decimal exponent of
13318 2. If the value is zero, the exponent shall be zero.

13319 A **double** argument representing an infinity or NaN shall be converted in the style of
13320 an `f` or `F` conversion specifier.

13321 `c` The **int** argument shall be converted to an **unsigned char**, and the resulting byte shall
13322 be written.

13323 If an `l` (ell) qualifier is present, the **wint_t** argument shall be converted as if by an `ls`
13324 conversion specification with no precision and an argument that points to a two-
13325 element array of type **wchar_t**, the first element of which contains the **wint_t** argument
13326 to the `ls` conversion specification and the second element contains a null wide
13327 character.

13328 `s` The argument shall be a pointer to an array of **char**. Bytes from the array shall be
13329 written up to (but not including) any terminating null byte. If the precision is specified,
13330 no more than that many bytes shall be written. If the precision is not specified or is
13331 greater than the size of the array, the application shall ensure that the array contains a
13332 null byte.

13333 If an `l` (ell) qualifier is present, the argument shall be a pointer to an array of type
13334 **wchar_t**. Wide characters from the array shall be converted to characters (each as if by
13335 a call to the *wcrtomb*( ) function, with the conversion state described by an **mbstate_t**
13336 object initialized to zero before the first wide character is converted) up to and
13337 including a terminating null wide character. The resulting characters shall be written
13338 up to (but not including) the terminating null character (byte). If no precision is
13339 specified, the application shall ensure that the array contains a null wide character. If a
13340 precision is specified, no more than that many characters (bytes) shall be written
13341 (including shift sequences, if any), and the array shall contain a null wide character if,
13342 to equal the character sequence length given by the precision, the function would need
13343 to access a wide character one past the end of the array. In no case shall a partial
13344 character be written.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`p` The argument shall be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.

`n` The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the *fprintf*( ) functions. No argument is converted.

XSI `C` Equivalent to `lc`.

XSI `S` Equivalent to `ls`.

`%` Print a `'%'` character; no argument is converted. The complete conversion specification shall be `%%`.

If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.

In no case shall a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by *fprintf*( ) and *printf*( ) are printed as if *fputc*( ) had been called.

For the `a` and `A` conversion specifiers, if FLT_RADIX is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.

For `a` and `A` conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

For the `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L <= D <= U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.

CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the call to a successful execution of *fprintf*( ) or *printf*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( ).

RETURN VALUE

Upon successful completion, the *fprintf*( ) and *printf*( ) functions shall return the number of bytes transmitted.

Upon successful completion, the *sprintf*( ) function shall return the number of bytes written to *s*, excluding the terminating null byte.

Upon successful completion, the *snprintf*( ) function shall return the number of bytes that would be written to *s* had *n* been sufficiently large excluding the terminating null byte.

If an output error was encountered, these functions shall return a negative value.

If the value of *n* is zero on a call to *snprintf*( ), nothing shall be written, the number of bytes that would have been written had *n* been sufficiently large excluding the terminating null shall be returned, and *s* may be a null pointer.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13388 **ERRORS**

13389 For the conditions under which *fprintf*( ) and *printf*( ) fail and may fail, refer to *fputc*( ) or
13390 *fputwc*( ).

13391 In addition, all forms of *fprintf*( ) may fail if:

13392 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been
13393 detected.

13394 XSI [EINVAL] There are insufficient arguments.

13395 The *printf*( ) and *fprintf*( ) functions may fail if:

13396 XSI [ENOMEM] Insufficient storage space is available.

13397 The *snprintf*( ) function shall fail if:

13398 XSI [EOVERFLOW] The value of *n* is greater than {INT_MAX} or the number of bytes needed to
13399 hold the output excluding the terminating null is greater than {INT_MAX}.

13400 **EXAMPLES**

13401 **Printing Language-Independent Date and Time**

13402 The following statement can be used to print date and time using a language-independent
13403 format:

13404
```
printf(format, weekday, month, day, hour, min);
```

13405 For American usage, *format* could be a pointer to the following string:

13406
```
"%s, %s %d, %d:%.2d\n"
```

13407 This example would produce the following message:

13408
```
Sunday, July 3, 10:02
```

13409 For German usage, *format* could be a pointer to the following string:

13410
```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

13411 This definition of *format* would produce the following message:

13412
```
Sonntag, 3. Juli, 10:02
```

13413 **Printing File Information**

13414 The following example prints information about the type, permissions, and number of links of a
13415 specific file in a directory.

13416 The first two calls to *printf*( ) use data decoded from a previous *stat*( ) call. The user-defined
13417 *strperm*( ) function shall return a string similar to the one at the beginning of the output for the
13418 following command:

13419
```
ls –l
```

13420 The next call to *printf*( ) outputs the owner's name if it is found using *getpwuid*( ); the *getpwuid*( )
13421 function shall return a **passwd** structure from which the name of the user is extracted. If the user
13422 name is not found, the program instead prints out the numeric value of the user ID.

13423 The next call prints out the group name if it is found using *getgrgid*( ); *getgrgid*( ) is very similar to
13424 *getpwuid*( ) except that it shall return group information based on the group number. Once
13425 again, if the group is not found, the program prints the numeric value of the group for the entry.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`--`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The final call to *printf*( ) prints the size of the file.

```
#include <stdio.h>
#include <sys/types.h>
#include <pwd.h>
#include <grp.h>

char *strperm (mode_t);
...
struct stat statbuf;
struct passwd *pwd;
struct group *grp;
...
printf("%10.10s", strperm (statbuf.st_mode));
printf("%4d", statbuf.st_nlink);

if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
    printf(" %-8.8s", pwd->pw_name);
else
    printf(" %-8ld", (long) statbuf.st_uid);

if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8ld", (long) statbuf.st_gid);

printf("%9jd", (intmax_t) statbuf.st_size);
...
```

Printing a Localized Date String

The following example gets a localized date string. The *nl_langinfo*( ) function shall return the localized date string, which specifies the order and layout of the date. The *strftime*( ) function takes this information and, using the **tm** structure for values, places the date and time information into *datestring*. The *printf*( ) function then outputs *datestring* and the name of the entry.

```
#include <stdio.h>
#include <time.h>
#include <langinfo.h>
...
struct dirent *dp;
struct tm *tm;
char datestring[256];
...
strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);

printf(" %s %s\n", datestring, dp->d_name);
...
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13466 **Printing Error Information**

13467 The following example uses *fprintf*( ) to write error information to standard error.

13468 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If
13469 the file already exists, this is an error, as indicated by the O_EXCL flag on the *open*( ) function. If
13470 the call fails, the program assumes that someone else is updating the password file, and the
13471 program exits.

13472 The next group of calls saves a new password file as the current password file by creating a link
13473 between **LOCKFILE** and the new password file **PASSWDFILE**.

```
13474 #include <sys/types.h>
13475 #include <sys/stat.h>
13476 #include <fcntl.h>
13477 #include <stdio.h>
13478 #include <stdlib.h>
13479 #include <unistd.h>
13480 #include <string.h>
13481 #include <errno.h>

13482 #define LOCKFILE "/etc/ptmp"
13483 #define PASSWDFILE "/etc/passwd"
13484 ...
13485 int pfd;
13486 ...
13487 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
13488     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
13489 {
13490     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
13491     exit(1);
13492 }
13493 ...
13494 if (link(LOCKFILE,PASSWDFILE) == -1) {
13495     fprintf(stderr, "Link error: %s\n", strerror(errno));
13496     exit(1);
13497 }
13498 ...
```

13499 **Printing Usage Information**

13500 The following example checks to make sure the program has the necessary arguments, and uses
13501 *fprintf*( ) to print usage information if the expected number of arguments is not present.

```
13502 #include <stdio.h>
13503 #include <stdlib.h>
13504 ...
13505 char *Options = "hdbtl";
13506 ...
13507 if (argc < 2) {
13508     fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
13509 }
13510 ...
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13511 **Formatting a Decimal String**

13512 The following example prints a key and data pair on *stdout*. Note use of the '*' (asterisk) in the
13513 format string; this ensures the correct number of decimal places for the element based on the
13514 number of elements requested.

```
13515 #include <stdio.h>
13516 ...
13517 long i;
13518 char *keystr;
13519 int elementlen, len;
13520 ...
13521 while (len < elementlen) {
13522 ...
13523     printf("%s Element%0*ld\n", keystr, elementlen, i);
13524 ...
13525 }
```

13526 **Creating a Filename**

13527 The following example creates a filename using information from a previous *getpwnam*()
13528 function that returned the HOME directory of the user.

```
13529 #include <stdio.h>
13530 #include <sys/types.h>
13531 #include <unistd.h>
13532 ...
13533 char filename[PATH_MAX+1];
13534 struct passwd *pw;
13535 ...
13536 sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
13537 ...
```

13538 **Reporting an Event**

13539 The following example loops until an event has timed out. The *pause*() function waits forever
13540 unless it receives a signal. The *fprintf*() statement should never occur due to the possible return
13541 values of *pause*().

```
13542 #include <stdio.h>
13543 #include <unistd.h>
13544 #include <string.h>
13545 #include <errno.h>
13546 ...
13547 while (!event_complete) {
13548 ...
13549     if (pause() != -1 || errno != EINTR)
13550         fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
13551 }
13552 ...
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Printing Monetary Information

The following example uses *strfmon*( ) to convert a number and store it as a formatted monetary string named *convbuf*. If the first number is printed, the program prints the format and the description; otherwise, it just prints the number.

```
#include <monetary.h>
#include <stdio.h>
...
struct tblfmt {
    char *format;
    char *description;
};

struct tblfmt table[] = {
    { "%n", "default formatting" },
    { "%11n", "right align within an 11 character field" },
    { "%#5n", "aligned columns for values up to 99 999" },
    { "%=*#5n", "specify a fill character" },
    { "%=0#5n", "fill characters do not use grouping" },
    { "%^#5n", "disable the grouping separator" },
    { "%^#5.0n", "round off to whole units" },
    { "%^#5.4n", "increase the precision" },
    { "%(#5n", "use an alternative pos/neg style" },
    { "%!(#5n", "disable the currency symbol" },
};
...
float input[3];
int i, j;
char convbuf[100];
...
strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);

if (j == 0) {
    printf("%s%s%s\n", table[i].format,
        convbuf, table[i].description);
}
else {
    printf("%s\n", convbuf);
}
...
```

Printing Wide Characters

The following example prints a series of wide characters. Suppose that `"L'@'"` expands to three bytes:

```
wchar_t wz [3] = L"@@";       // Zero-terminated
wchar_t wn [3] = L"@@@";      // Unterminated

fprintf (stdout,"%ls", wz);   // Outputs 6 bytes
fprintf (stdout,"%ls", wn);   // Undefined because wn has no terminator
fprintf (stdout,"%4ls", wz);  // Outputs 3 bytes
fprintf (stdout,"%4ls", wn);  // Outputs 3 bytes; no terminator needed
fprintf (stdout,"%9ls", wz);  // Outputs 6 bytes
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
fprintf (stdout,"%9ls", wn);  // Outputs 9 bytes; no terminator needed
fprintf (stdout,"%10ls", wz); // Outputs 6 bytes
fprintf (stdout,"%10ls", wn); // Undefined because wn has no terminator
```

In the last line of the example, after processing three characters, nine bytes have been output. The fourth character must then be examined to determine whether it converts to one byte or more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth character in the array, the behavior is undefined.

APPLICATION USAGE

If the application calling *fprintf*( ) has any objects of type **wint_t** or **wchar_t**, it must also include the **<wchar.h>** header to have these objects defined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fputc( ), *fscanf*( ), *setlocale*( ), *strfmon*( ), *wcrtomb*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier can now be used with `c` and `s` conversion specifiers.

The *snprintf*( ) function is new in Issue 5.

Issue 6

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fprintf*( ), *printf*( ), *snprintf*( ), and *sprintf*( ) are updated, and the XSI shading is removed from *snprintf*( ).
- The description of *snprintf*( ) is aligned with the ISO C standard. Note that this supersedes the *snprintf*( ) description in The Open Group Base Resolution bwg98-006, which changed the behavior from Issue 5.
- The DESCRIPTION is updated.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ consistently.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

An example of printing wide characters is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13637 **NAME**

13638 fputc — put a byte on a stream

13639 **SYNOPSIS**

13640 `#include <stdio.h>`

13641 `int fputc(int c, FILE *stream);`

13642 **DESCRIPTION**

13643 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13644 conflict between the requirements described here and the ISO C standard is unintentional. This
13645 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13646 The *fputc*( ) function shall write the byte specified by *c* (converted to an **unsigned char**) to the
13647 output stream pointed to by *stream*, at the position indicated by the associated file-position
13648 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file
13649 cannot support positioning requests, or if the stream was opened with append mode, the byte
13650 shall be appended to the output stream.

13651 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
13652 execution of *fputc*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same
13653 stream or a call to *exit*( ) or *abort*( ).

13654 **RETURN VALUE**

13655 Upon successful completion, *fputc*( ) shall return the value it has written. Otherwise, it shall
13656 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the
13657 error.

13658 **ERRORS**

13659 The *fputc*( ) function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be
13660 flushed, and:

13661 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
13662 process would be delayed in the write operation.

13663 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
13664 writing.

13665 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

13666 XSI [EFBIG] An attempt was made to write to a file that exceeds the process' file size limit.

13667 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
13668 offset maximum.

13669 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
13670 was transferred.

13671 CX [EIO] A physical I/O error has occurred, or the process is a member of a
13672 background process group attempting to write to its controlling terminal,
13673 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
13674 process group of the process is orphaned. This error may also be returned
13675 under implementation-defined conditions.

13676 CX [ENOSPC] There was no free space remaining on the device containing the file.

13677 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
13678 any process. A SIGPIPE signal shall also be sent to the thread.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *fputc*( ) function may fail if:

CX [ENOMEM] Insufficient storage space is available.

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ferror( ), *fopen*( ), *getrlimit*( ), *putc*( ), *puts*( ), *setbuf*( ), *ulimit*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EFBIG] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13704 **NAME**

13705 fputs — put a string on a stream

13706 **SYNOPSIS**

13707 ```
#include <stdio.h>
```

13708 ```
int fputs(const char *restrict s, FILE *restrict stream);
```

13709 **DESCRIPTION**

13710 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13711 conflict between the requirements described here and the ISO C standard is unintentional. This
13712 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13713 The *fputs*( ) function shall write the null-terminated string pointed to by *s* to the stream pointed
13714 to by *stream*. The terminating null byte shall not be written.

13715 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
13716 execution of *fputs*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same
13717 stream or a call to *exit*( ) or *abort*( ).

13718 **RETURN VALUE**

13719 Upon successful completion, *fputs*( ) shall return a non-negative number. Otherwise, it shall
13720 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

13721 **ERRORS**

13722 Refer to *fputc*( ).

13723 **EXAMPLES**

13724 **Printing to Standard Output**

13725 The following example gets the current time, converts it to a string using *localtime*( ) and
13726 *asctime*( ), and prints it to standard output using *fputs*( ). It then prints the number of minutes to
13727 an event for which it is waiting.

```
13728 #include <time.h>
13729 #include <stdio.h>
13730 ...
13731 time_t now;
13732 int minutes_to_event;
13733 ...
13734 time(&now);
13735 printf("The time is ");
13736 fputs(asctime(localtime(&now)), stdout);
13737 printf("There are still %d minutes to the event.\n",
13738     minutes_to_event);
13739 ...
```

13740 **APPLICATION USAGE**

13741 The *puts*( ) function appends a <newline> while *fputs*( ) does not.

13742 **RATIONALE**

13743 None.

13744 **FUTURE DIRECTIONS**

13745 None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13746 **SEE ALSO**

13747 *fopen*( ), *putc*( ), *puts*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

13748 **CHANGE HISTORY**

13749 First released in Issue 1. Derived from Issue 1 of the SVID.

13750 **Issue 6**

13751 Extensions beyond the ISO C standard are marked.

13752 The *fputs*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

13753 **NAME**

13754 fputwc — put a wide-character code on a stream

13755 **SYNOPSIS**

13756 ```
#include <stdio.h>
```

13757 ```
#include <wchar.h>
```

13758 ```
wint_t fputwc(wchar_t wc, FILE *stream);
```

13759 **DESCRIPTION**

13760 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13761 conflict between the requirements described here and the ISO C standard is unintentional. This
13762 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13763 The *fputwc*( ) function shall write the character corresponding to the wide-character code *wc* to
13764 the output stream pointed to by *stream*, at the position indicated by the associated file-position
13765 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot
13766 support positioning requests, or if the stream was opened with append mode, the character is
13767 appended to the output stream. If an error occurs while writing the character, the shift state of
13768 the output file is left in an undefined state.

13769 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
13770 execution of *fputwc*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the
13771 same stream or a call to *exit*( ) or *abort*( ).

13772 **RETURN VALUE**

13773 Upon successful completion, *fputwc*( ) shall return *wc*. Otherwise, it shall return WEOF, the error
13774 CX indicator for the stream shall be set, and *errno* shall be set to indicate the error.

13775 **ERRORS**

13776 The *fputwc*( ) function shall fail if either the stream is unbuffered or data in the *stream*'s buffer
13777 needs to be written, and:

13778 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
13779 process would be delayed in the write operation.

13780 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
13781 writing.

13782 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size or
13783 the process' file size limit.

13784 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
13785 offset maximum associated with the corresponding stream.

13786 [EILSEQ] The wide-character code *wc* does not correspond to a valid character.

13787 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
13788 was transferred.

13789 CX [EIO] A physical I/O error has occurred, or the process is a member of a
13790 background process group attempting to write to its controlling terminal,
13791 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
13792 process group of the process is orphaned. This error may also be returned
13793 under implementation-defined conditions.

13794 CX [ENOSPC] There was no free space remaining on the device containing the file.

13795 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
13796 any process. A SIGPIPE signal shall also be sent to the thread.


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale
```

The *fputwc*( ) function may fail if:

CX [ENOMEM] Insufficient storage space is available.

CX [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ferror( ), *fopen*( ), *setbuf*( ), *ulimit*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

The Optional Header (OH) marking is removed from **<stdio.h>**.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] and [EIO] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13825 **NAME**

13826 fputws — put a wide-character string on a stream

13827 **SYNOPSIS**

13828 `#include <stdio.h>`

13829 `#include <wchar.h>`

13830 `int fputws(const wchar_t *restrict ws, FILE *restrict stream);`

13831 **DESCRIPTION**

13832 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13833 conflict between the requirements described here and the ISO C standard is unintentional. This
13834 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13835 The *fputws*( ) function shall write a character string corresponding to the (null-terminated)
13836 wide-character string pointed to by *ws* to the stream pointed to by *stream*. No character
13837 corresponding to the terminating null wide-character code shall be written.

13838 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
13839 execution of *fputws*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the
13840 same stream or a call to *exit*( ) or *abort*( ).

13841 **RETURN VALUE**

13842 Upon successful completion, *fputws*( ) shall return a non-negative number. Otherwise, it shall
13843 CX return –1, set an error indicator for the stream, and set *errno* to indicate the error.

13844 **ERRORS**

13845 Refer to *fputwc*( ).

13846 **EXAMPLES**

13847 None.

13848 **APPLICATION USAGE**

13849 The *fputws*( ) function does not append a <newline>.

13850 **RATIONALE**

13851 None.

13852 **FUTURE DIRECTIONS**

13853 None.

13854 **SEE ALSO**

13855 *fopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<wchar.h>**

13856 **CHANGE HISTORY**

13857 First released in Issue 4. Derived from the MSE working draft.

13858 **Issue 5**

13859 The Optional Header (OH) marking is removed from **<stdio.h>**.

13860 **Issue 6**

13861 Extensions beyond the ISO C standard are marked.

13862 The *fputws*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fread — binary input

SYNOPSIS

```
#include <stdio.h>

size_t fread(void *restrict ptr, size_t size, size_t nitems,
    FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fread*( ) function shall read into the array pointed to by *ptr* up to *nitems* elements whose size is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fgetc*( ) function and the results stored, in the order read, in an array of **unsigned char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be advanced by the number of bytes successfully read. If an error occurs, the resulting value of the file position indicator for the stream is unspecified. If a partial element is read, its value is unspecified.

CX The *fread*( ) function may mark the *st_atime* field of the file associated with *stream* for update. The *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fgetwc*( ), *fgetws*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *gets*( ), or *scanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ) or *ungetwc*( ).

RETURN VALUE

Upon successful completion, *fread*( ) shall return the number of elements successfully read which is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread*( ) shall return 0 and the contents of the array and the state of the stream remain unchanged.

CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

ERRORS

Refer to *fgetc*( ).

EXAMPLES

Reading from a Stream

The following example reads a single element from the *fp* stream into the array pointed to by *buf*.

```
#include <stdio.h>
...
size_t bytes_read;
char buf[100];
FILE *fp;
...
bytes_read = fread(buf, sizeof(buf), 1, fp);
...
```

APPLICATION USAGE

The *ferror*( ) or *feof*( ) functions must be used to distinguish between an error condition and an end-of-file condition.

Because of possible differences in element length and byte ordering, files written using *fwrite*( ) are application-dependent, and possibly cannot be read using *fread*( ) by a different application

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

or by the same application on a different processor.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feof( ), *ferror*( ), *fgetc*( ), *fopen*( ), *getc*( ), *gets*( ), *scanf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following changes are made for alignment with the ISO / IEC 9899: 1999 standard:

- The *fread*( ) prototype is updated.
- The DESCRIPTION is updated to describe how the bytes from a call to *fgetc*( ) are stored.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

free — free allocated memory

SYNOPSIS

```
#include <stdlib.h>

void free(void *ptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *free*( ) function shall cause the space pointed to by *ptr* to be deallocated; that is, made available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the ADV argument does not match a pointer earlier returned by the *calloc*( ), *malloc*( ), *posix_memalign*( ), XSI *realloc*( ), or *strdup*( ) function, or if the space has been deallocated by a call to *free*( ) or *realloc*( ), the behavior is undefined.

Any use of a pointer that refers to freed space results in undefined behavior.

RETURN VALUE

The *free*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

There is now no requirement for the implementation to support the inclusion of **<malloc.h>**.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

calloc( ), *malloc*( ), *realloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Reference to the *valloc*( ) function is removed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13956 **NAME**

13957 freeaddrinfo, getaddrinfo — get address information

13958 **SYNOPSIS**

13959 ```
#include <sys/socket.h>
13960 #include <netdb.h>

13961 void freeaddrinfo(struct addrinfo *ai);
13962 int getaddrinfo(const char *restrict nodename,
13963   const char *restrict servname,
13964   const struct addrinfo *restrict hints,
13965   struct addrinfo **restrict res);
```

13966 **DESCRIPTION**

13967 The *freeaddrinfo*( ) function shall free one or more **addrinfo** structures returned by *getaddrinfo*( ),
13968 along with any additional storage associated with those structures. If the *ai_next* field of the
13969 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo*( ) function shall
13970 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo*( ).

13971 The *getaddrinfo*( ) function shall translate the name of a service location (for example, a host
13972 name) and/or a service name and shall return a set of socket addresses and associated
13973 information to be used in creating a socket with which to address the specified service.

13974 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,
13975 RFC 1035, and RFC 1886.

13976 The *freeaddrinfo*( ) and *getaddrinfo*( ) functions shall be thread-safe.

13977 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated
13978 strings. One or both of these two arguments shall be supplied by the application as a non-null
13979 pointer.

13980 The format of a valid name depends on the address family or families. If a specific family is not
13981 given and the name could be interpreted as valid within multiple supported families, the
13982 implementation shall attempt to resolve the name in all supported families and, in absence of
13983 errors, one or more results shall be returned.

13984 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If
13985 IP6 the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, valid descriptive names
13986 include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings
13987 using Internet standard dot notation as specified in *inet_addr*( ) are valid.

13988 IP6 If the specified address family is AF_INET6 or AF_UNSPEC, standard IPv6 text forms described
13989 in *inet_ntop*( ) are valid.

13990 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the
13991 requested service location is local to the caller.

13992 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If
13993 *servname* is not null, it is a null-terminated character string identifying the requested service. This
13994 can be either a descriptive name or a numeric representation suitable for use with the address
13995 IP6 family or families. If the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, the
13996 service can be specified as a string specifying a decimal port number.

13997 If the *hints* argument is not null, it refers to a structure containing input values that may direct
13998 the operation by providing options and by limiting the returned information to a specific socket
13999 type, address family, and/or protocol. In this *hints* structure every member other than *ai_flags*,
14000 *ai_family*, *ai_socktype*, and *ai_protocol* shall be set to zero or a null pointer. A value of
14001 AF_UNSPEC for *ai_family* means that the caller shall accept any address family. A value of zero


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
```

for *ai_socktype* means that the caller shall accept any socket type. A value of zero for *ai_protocol* means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if it referred to a structure containing the value zero for the *ai_flags*, *ai_socktype*, and *ai_protocol* fields, and AF_UNSPEC for the *ai_family* field.

The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG.

If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in binding a socket for accepting incoming connections for the specified service. In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address. If the AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to *connect*( ) (for a connection-mode protocol) or for a call to *connect*( ), *sendto*( ), or *sendmsg*( ) (for a connectionless protocol). In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to the loopback address. The AI_PASSIVE flag shall be ignored if the *nodename* argument is not null.

If the AI_CANONNAME flag is specified and the *nodename* argument is not null, the function shall attempt to determine the canonical name corresponding to *nodename* (for example, if *nodename* is an alias or shorthand notation for a complete name).

Note: Since different implementations use different conceptual models, the terms ''canonical name'' and ''alias'' cannot be precisely defined for the general case. However, Domain Name System implementations are expected to interpret them as they are used in RFC 1034.

A numeric host address string is not a ''name'', and thus does not have a ''canonical name'' form; no address to host name translation is performed. See below for handling of the case where a canonical name cannot be obtained.

If the AI_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a numeric host address string. Otherwise, an [EAI_NONAME] error is returned. This flag shall prevent any type of name resolution service (for example, the DNS) from being invoked.

If the AI_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a numeric port string. Otherwise, an [EAI_NONAME] error shall be returned. This flag shall prevent any type of name resolution service (for example, NIS+) from being invoked.

IP6 If the AI_V4MAPPED flag is specified along with an *ai_family* of AF_INET6, then *getaddrinfo*( ) shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai_addrlen* shall be 16). The AI_V4MAPPED flag shall be ignored unless *ai_family* equals AF_INET6. If the AI_ALL flag is used with the AI_V4MAPPED flag, then *getaddrinfo*( ) shall return all matching IPv6 and IPv4 addresses. The AI_ALL flag without the AI_V4MAPPED flag is ignored.

If the AI_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4 IP6 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6 address is configured on the local system.

The *ai_socktype* field to which argument *hints* points specifies the socket type for the service, as defined in *socket*( ). If a specific socket type is not given (for example, a value of zero) and the service name could be interpreted as valid with multiple supported socket types, the implementation shall attempt to resolve the service name for all supported socket types and, in the absence of errors, all possible results shall be returned. A non-zero socket type value shall limit the returned information to values with the specified socket type.

If the *ai_family* field to which *hints* points has the value AF_UNSPEC, addresses shall be returned for use with any address family that can be used with the specified *nodename* and/or *servname*. Otherwise, addresses shall be returned for use only with the specified address family.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If *ai_family* is not AF_UNSPEC and *ai_protocol* is not zero, then addresses are returned for use only with the specified address family and protocol; the value of *ai_protocol* shall be interpreted as in a call to the *socket*( ) function with the corresponding values of *ai_family* and *ai_protocol*.

RETURN VALUE

A zero return value for *getaddrinfo*( ) indicates successful completion; a non-zero return value indicates failure. The possible values for the failures are listed in the ERRORS section.

Upon successful return of *getaddrinfo*( ), the location to which *res* points shall refer to a linked list of **addrinfo** structures, each of which shall specify a socket address and information for use in creating a socket with which to use that socket address. The list shall include at least one **addrinfo** structure. The *ai_next* field of each structure contains a pointer to the next structure on the list, or a null pointer if it is the last structure on the list. Each structure on the list shall include values for use with a call to the *socket*( ) function, and a socket address for use with the *connect*( ) function or, if the AI_PASSIVE flag was specified, for use with the *bind*( ) function. The fields *ai_family*, *ai_socktype*, and *ai_protocol* shall be usable as the arguments to the *socket*( ) function to create a socket suitable for use with the returned address. The fields *ai_addr* and *ai_addrlen* are usable as the arguments to the *connect*( ) or *bind*( ) functions with such a socket, according to the AI_PASSIVE flag.

If *nodename* is not null, and if requested by the AI_CANONNAME flag, the *ai_canonname* field of the first returned **addrinfo** structure shall point to a null-terminated string containing the canonical name corresponding to the input *nodename*; if the canonical name is not available, then *ai_canonname* shall refer to the *nodename* argument or a string with the same contents. The contents of the *ai_flags* field of the returned structures are undefined.

All fields in socket address structures returned by *getaddrinfo*( ) that are not filled in through an explicit argument (for example, *sin6_flowinfo*) shall be set to zero.

Note: This makes it easier to compare socket address structures.

ERRORS

The *getaddrinfo*( ) function shall fail and return the corresponding value if:

[EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

[EAI_BADFLAGS] The *flags* parameter had an invalid value.

[EAI_FAIL] A non-recoverable error occurred when attempting to resolve the name.

[EAI_FAMILY] The address family was not recognized.

[EAI_MEMORY] There was a memory allocation failure when trying to allocate storage for the return value.

[EAI_NONAME] The name does not resolve for the supplied parameters.

Neither *nodename* nor *servname* were supplied. At least one of these shall be supplied.

[EAI_SERVICE] The service passed was not recognized for the specified socket type.

[EAI_SOCKTYPE] The intended socket type was not recognized.

[EAI_SYSTEM] A system error occurred; the error code can be found in *errno*.

[EAI_OVERFLOW] An argument buffer overflowed.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

If the caller handles only TCP and not UDP, for example, then the *ai_protocol* member of the *hints* structure should be set to IPPROTO_TCP when *getaddrinfo*( ) is called.

If the caller handles only IPv4 and not IPv6, then the *ai_family* member of the *hints* structure should be set to AF_INET when *getaddrinfo*( ) is called.

The term ‘‘canonical name’’ is misleading; it is taken from the Domain Name System (RFC 2181). It should be noted that the canonical name is a result of alias processing, and not necessarily a unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this in the Domain Name System context.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

connect( ), *gai_strerror*( ), *gethostbyaddr*( ), *getnameinfo*( ), *getservbyname*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<netdb.h>**, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *getaddrinfo*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the DESCRIPTION and adding text to the APPLICATION USAGE related to the term ‘‘canonical name’’. A reference to RFC 2181 is also added to the Informative References.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for alignment with IPv6. These include the following:

- Adding AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG to the allowed values for the *ai_flags* field
- Adding a description of AI_ADDRCONFIG
- Adding a description of the consequences of ignoring the AI_PASSIVE flag.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

freopen — open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *freopen(const char *restrict filename, const char *restrict mode,
    FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *freopen*( ) function shall first attempt to flush the stream and close any file descriptor associated with *stream*. Failure to flush or close the file descriptor successfully shall be ignored. The error and end-of-file indicators for the stream shall be cleared.

The *freopen*( ) function shall open the file whose pathname is the string pointed to by *filename* and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in *fopen*( ).

The original stream shall be closed regardless of whether the subsequent open succeeds.

If *filename* is a null pointer, the *freopen*( ) function shall attempt to change the mode of the stream to that specified by *mode*, as if the name of the file currently associated with the stream had been used. It is implementation-defined which changes of mode are permitted (if any), and under what circumstances.

XSI After a successful call to the *freopen*( ) function, the orientation of the stream shall be cleared, the encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an initial conversion state.

CX The largest value that can be represented correctly in an object of type **off_t** shall be established as the offset maximum in the open file description.

RETURN VALUE

Upon successful completion, *freopen*( ) shall return the value of *stream*. Otherwise, a null pointer shall be returned, CX and *errno* shall be set to indicate the error.

ERRORS

The *freopen*( ) function shall fail if:

CX [EACCES] Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *mode* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

CX [EINTR] A signal was caught during *freopen*( ).

CX [EISDIR] The named file is a directory and *mode* requires write access.

CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

CX [ENAMETOOLONG] The length of the *filename* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

14167 CX [ENFILE] The maximum allowable number of files is currently open in the system.

14168 CX [ENOENT] A component of *filename* does not name an existing file or *filename* is an empty
14169 string.

14170 CX [ENOSPC] The directory or file system that would contain the new file cannot be
14171 expanded, the file does not exist, and it was to be created.

14172 CX [ENOTDIR] A component of the path prefix is not a directory.

14173 CX [ENXIO] The named file is a character special or block special file, and the device
14174 associated with this special file does not exist.

14175 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented
14176 correctly in an object of type **off_t**.

14177 CX [EROFS] The named file resides on a read-only file system and *mode* requires write
14178 access.

14179 The *freopen*( ) function may fail if:

14180 CX [EINVAL] The value of the *mode* argument is not valid.

14181 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
14182 resolution of the *path* argument.

14183 CX [ENAMETOOLONG]
14184 Pathname resolution of a symbolic link produced an intermediate result
14185 whose length exceeds {PATH_MAX}.

14186 CX [ENOMEM] Insufficient storage space is available.

14187 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
14188 capabilities of the device.

14189 CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode*
14190 requires write access.

14191 **EXAMPLES**

14192 **Directing Standard Output to a File**

14193 The following example logs all standard output to the **/tmp/logfile** file.

```
14194 #include <stdio.h>
14195 ...
14196 FILE *fp;
14197 ...
14198 fp = freopen ("/tmp/logfile", "a+", stdout);
14199 ...
```

14200 **APPLICATION USAGE**

14201 The *freopen*( ) function is typically used to attach the preopened *streams* associated with *stdin*,
14202 *stdout*, and *stderr* to other files.

14203 **RATIONALE**

14204 None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

fclose( ), *fopen*( ), *fdopen*( ), *mbsinit*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the conversion state of the stream is set to an initial conversion state by a successful call to the *freopen*( ) function.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

The following changes are made for alignment with the ISO / IEC 9899: 1999 standard:

- The *freopen*( ) prototype is updated.
- The DESCRIPTION is updated.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

The DESCRIPTION is updated regarding failure to close, changing the ‘‘file’’ to ‘‘file descriptor’’.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

SYNOPSIS

```
#include <math.h>

double frexp(double num, int *exp);
float frexpf(float num, int *exp);
long double frexpl(long double num, int *exp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall break a floating-point number *num* into a normalized fraction and an integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.

RETURN VALUE

For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the interval [½,1) or 0, and *num* equals *x* times 2 raised to the power **exp*.

MX If *num* is NaN, a NaN shall be returned, and the value of **exp* is unspecified.

If *num* is ±0, ±0 shall be returned, and the value of **exp* shall be 0.

If *num* is ±Inf, *num* shall be returned, and the value of **exp* is unspecified.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isnan( ), *ldexp*( ), *modf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *frexpf*( ) and *frexpl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fscanf, scanf, sscanf — convert formatted input

SYNOPSIS

```
#include <stdio.h>

int fscanf(FILE *restrict stream, const char *restrict format, ... );
int scanf(const char *restrict format, ... );
int sscanf(const char *restrict s, const char *restrict format, ... );
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fscanf*( ) function shall read from the named input *stream*. The *scanf*( ) function shall read from the standard input stream *stdin*. The *sscanf*( ) function shall read from the string *s*. Each function reads bytes, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be evaluated but otherwise ignored.

XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character `%` (see below) is replaced by the sequence `"%`*n*`$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages. In format strings containing the `"%`*n*`$"` form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.

The *format* can contain either form of a conversion specification—that is, `%` or `"%`*n*`$"`—but the two forms cannot be mixed within a single *format* string. The only exception to this is that `%%` or `%*` can be mixed with the `"%`*n*`$"` form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are pointers.

CX The *fscanf*( ) function in all its forms shall allow detection of a language-dependent radix character in the input string. The radix character is defined in the program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period (`'.'`).

The format is a character string, beginning and ending in its initial shift state, if any, composed of zero or more directives. Each directive is composed of one of the following: one or more white-space characters (<space>s, <tab>s, <newline>s, <vertical-tab>s, or <form-feed>s); an ordinary character (neither `'%'` nor a white-space character); or a conversion specification. Each
XSI conversion specification is introduced by the character `'%'` or the character sequence `"%`*n*`$"`, after which the following appear in sequence:

- An optional assignment-suppressing character `'*'`.
- An optional non-zero decimal integer that specifies the maximum field width.
- An option length modifier that specifies the size of the receiving object.
- A *conversion specifier* character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *fscanf*( ) functions shall execute each directive of the format in turn. If a directive fails, as detailed below, the function shall return. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space characters shall be executed by reading input until no more valid input can be read, or up to the first byte which is not a white-space character, which remains unread.

A directive that is an ordinary character shall be executed as follows: the next byte shall be read from the input and compared with the byte that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a character from being read, the directive shall fail.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion character. A conversion specification shall be executed in the following steps.

Input white-space characters (as specified by *isspace*( )) shall be skipped, unless the conversion specification includes a `[`, `c`, `C`, or `n` conversion specifier.

An item shall be read from the input, unless the conversion specification includes an `n` conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to any specified maximum field width, which may be measured in characters or bytes dependent on the conversion specifier) which is an initial subsequence of a matching sequence. The first byte, if any, after the input item shall remain unread. If the length of the input item is 0, the execution of the conversion specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion specification, the count of input bytes) shall be converted to a type appropriate to the conversion character. If the input item is not a matching sequence, the execution of the conversion specification fails; this condition is a matching failure. Unless assignment suppression was indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if XSI the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the character sequence `"%`*n*`$"`. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The length modifiers and their meanings are:

`hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **signed char** or **unsigned char**.

`h` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **short** or **unsigned short**.

`l` (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **long** or **unsigned long**; that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to an argument with type pointer to **double**; or that a following `c`, `s`, or `[` conversion specifier applies to an argument with type pointer to **wchar_t**.

`ll` (ell-ell)
Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an argument with type pointer to **long long** or **unsigned long long**.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14368 `j` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
14369 argument with type pointer to **intmax_t** or **uintmax_t**.

14370 `z` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
14371 argument with type pointer to **size_t** or the corresponding signed integer type.

14372 `t` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
14373 argument with type pointer to **ptrdiff_t** or the corresponding **unsigned** type.

14374 `L` Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to an
14375 argument with type pointer to **long double**.

14376 If a length modifier appears with any conversion specifier other than as specified above, the
14377 behavior is undefined.

14378 The following conversion specifiers are valid:

14379 `d` Matches an optionally signed decimal integer, whose format is the same as expected for
14380 the subject sequence of *strtol*( ) with the value 10 for the *base* argument. In the absence
14381 of a size modifier, the application shall ensure that the corresponding argument is a
14382 pointer to **int**.

14383 `i` Matches an optionally signed integer, whose format is the same as expected for the
14384 subject sequence of *strtol*( ) with 0 for the *base* argument. In the absence of a size
14385 modifier, the application shall ensure that the corresponding argument is a pointer to
14386 **int**.

14387 `o` Matches an optionally signed octal integer, whose format is the same as expected for
14388 the subject sequence of *strtoul*( ) with the value 8 for the *base* argument. In the absence
14389 of a size modifier, the application shall ensure that the corresponding argument is a
14390 pointer to **unsigned**.

14391 `u` Matches an optionally signed decimal integer, whose format is the same as expected for
14392 the subject sequence of *strtoul*( ) with the value 10 for the *base* argument. In the absence
14393 of a size modifier, the application shall ensure that the corresponding argument is a
14394 pointer to **unsigned**.

14395 `x` Matches an optionally signed hexadecimal integer, whose format is the same as
14396 expected for the subject sequence of *strtoul*( ) with the value 16 for the *base* argument. In
14397 the absence of a size modifier, the application shall ensure that the corresponding
14398 argument is a pointer to **unsigned**.

14399 `a`, `e`, `f`, `g`
14400 Matches an optionally signed floating-point number, infinity, or NaN, whose format is
14401 the same as expected for the subject sequence of *strtod*( ). In the absence of a size
14402 modifier, the application shall ensure that the corresponding argument is a pointer to
14403 **float**.

14404 If the *fprintf*( ) family of functions generates character string representations for infinity
14405 and NaN (a symbolic entity encoded in floating-point format) to support
14406 IEEE Std 754-1985, the *fscanf*( ) family of functions shall recognize them as input.

14407 `s` Matches a sequence of bytes that are not white-space characters. The application shall
14408 ensure that the corresponding argument is a pointer to the initial byte of an array of
14409 **char**, **signed char**, or **unsigned char** large enough to accept the sequence and a
14410 terminating null character code, which shall be added automatically.

14411 If an `l` (ell) qualifier is present, the input is a sequence of characters that begins in the
14412 initial shift state. Each character shall be converted to a wide character as if by a call to

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted. The application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide character, which shall be added automatically.

`[` Matches a non-empty sequence of bytes from a set of expected bytes (the *scanset*). The normal skip over white-space characters shall be suppressed in this case. The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned char** large enough to accept the sequence and a terminating null byte, which shall be added automatically.

If an `l` (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character in the sequence shall be converted to a wide character as if by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted. The application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide character, which shall be added automatically.

The conversion specification includes all subsequent bytes in the *format* string up to and including the matching right square bracket (`']'`). The bytes between the square brackets (the *scanlist*) comprise the scanset, unless the byte after the left square bracket is a circumflex (`'^'`), in which case the scanset contains all bytes that do not appear in the scanlist between the circumflex and the right square bracket. If the conversion specification begins with `"[]"` or `"[^]"`, the right square bracket is included in the scanlist and the next right square bracket is the matching right square bracket that ends the conversion specification; otherwise, the first right square bracket is the one that ends the conversion specification. If a `'-'` is in the scanlist and is not the first character, nor the second where the first character is a `'^'`, nor the last character, the behavior is implementation-defined.

`c` Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned char** large enough to accept the sequence. No null byte is added. The normal skip over white-space characters shall be suppressed in this case.

If an `l` (ell) qualifier is present, the input shall be a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted. The application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the resulting sequence of wide characters. No null wide character is added.

`p` Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the `%p` conversion specification of the corresponding *fprintf*() functions. The application shall ensure that the corresponding argument is a pointer to a pointer to **void**. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the `%p` conversion specification is undefined.

`n` No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

input so far by this call to the *fscanf*( ) functions. Execution of a `%n` conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.

XSI `C` Equivalent to `lc`.

XSI `S` Equivalent to `ls`.

`%` Matches a single `'%'` character; no conversion or assignment occurs. The complete conversion specification shall be `%%`.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers `A`, `E`, `F`, `G`, and `X` are also valid and shall be equivalent to `a`, `e`, `f`, `g`, and `x`, respectively.

If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for `%n`) have been read (other than leading white-space characters, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in *sscanf*( ) shall be equivalent to encountering end-of-file for *fscanf*( ).

If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white space (including <newline>s) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the `%n` conversion specification.

CX The *fscanf*( ) and *scanf*( ) functions may mark the *st_atime* field of the file associated with *stream* for update. The *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ), *fread*( ), *getc*( ), *getchar*( ), *gets*( ), *fscanf*( ), or *fscanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ).

RETURN VALUE

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF shall be returned. If a read CX error occurs, the error indicator for the stream is set, EOF shall be returned, and *errno* shall be set to indicate the error.

ERRORS

For the conditions under which the *fscanf*( ) functions fail and may fail, refer to *fgetc*( ) or *fgetwc*( ).

In addition, *fscanf*( ) may fail if:

XSI [EILSEQ] Input byte sequence does not form a valid character.

XSI [EINVAL] There are insufficient arguments.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string `"Hamster"`.

The call:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string `"56\0"` in *name*. The next call to *getchar*( ) shall return the character `'a'`.

Reading Data into an Array

The following call uses *fscanf*( ) to read three floating-point numbers from standard input into the *input* array.

```
float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

APPLICATION USAGE

If the application calling *fscanf*( ) has any objects of type **wint_t** or **wchar_t**, it must also include the **<wchar.h>** header to have these objects defined.

RATIONALE

This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few ''obvious'' things were not included. Specifically, the set of characters allowed in a scanset is limited to single-byte characters. In other similar places, multi-byte characters have been permitted, but for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications needing this could use the corresponding wide-character functions to achieve the desired results.

FUTURE DIRECTIONS

None.

SEE ALSO

getc( ), *printf*( ), *setlocale*( ), *strtod*( ), *strtol*( ), *strtoul*( ), *wcrtomb*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier is now defined for the `c`, `s`, and `[` conversion specifiers.

The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the *fprintf*( ) family of functions, then they are recognized by the *fscanf*( ) family.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several occurrences of ‘‘characters’’ in the text which have been replaced with the term ‘‘bytes’’.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fscanf*( ), *scanf*( ), and *sscanf*( ) are updated.
- The DESCRIPTION is updated.
- The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
- The `a`, `A`, and `F` conversion characters are added.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ consistently.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14554 **NAME**

14555 fseek, fseeko — reposition a file-position indicator in a stream

14556 **SYNOPSIS**

14557 ```
#include <stdio.h>
```

14558 ```
int fseek(FILE *stream, long offset, int whence);
```

14559 CX ```
int fseeko(FILE *stream, off_t offset, int whence);
```

14560

14561 **DESCRIPTION**

14562 CX The functionality described on this reference page is aligned with the ISO C standard. Any
14563 conflict between the requirements described here and the ISO C standard is unintentional. This
14564 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14565 The *fseek*( ) function shall set the file-position indicator for the stream pointed to by *stream*. If a
14566 read or write error occurs, the error indicator for the stream shall be set and *fseek*( ) fails.

14567 The new position, measured in bytes from the beginning of the file, shall be obtained by adding
14568 *offset* to the position specified by *whence*. The specified point is the beginning of the file for
14569 SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for
14570 SEEK_END.

14571 If the stream is to be used with wide-character input/output functions, the application shall
14572 ensure that *offset* is either 0 or a value returned by an earlier call to *ftell*( ) on the same stream and
14573 *whence* is SEEK_SET.

14574 A successful call to *fseek*( ) shall clear the end-of-file indicator for the stream and undo any effects
14575 of *ungetc*( ) and *ungetwc*( ) on the same stream. After an *fseek*( ) call, the next operation on an
14576 update stream may be either input or output.

14577 CX If the most recent operation, other than *ftell*( ), on a given stream is *fflush*( ), the file offset in the
14578 underlying open file description shall be adjusted to reflect the location specified by *fseek*( ).

14579 The *fseek*( ) function shall allow the file-position indicator to be set beyond the end of existing
14580 data in the file. If data is later written at this point, subsequent reads of data in the gap shall
14581 return bytes with the value 0 until data is actually written into the gap.

14582 The behavior of *fseek*( ) on devices which are incapable of seeking is implementation-defined.
14583 The value of the file offset associated with such a device is undefined.

14584 If the stream is writable and buffered data had not been written to the underlying file, *fseek*( )
14585 shall cause the unwritten data to be written to the file and shall mark the *st_ctime* and *st_mtime*
14586 fields of the file for update.

14587 In a locale with state-dependent encoding, whether *fseek*( ) restores the stream's shift state is
14588 implementation-defined.

14589 The *fseeko*( ) function shall be equivalent to the *fseek*( ) function except that the *offset* argument is
14590 of type **off_t**.

14591 **RETURN VALUE**

14592 CX The *fseek*( ) and *fseeko*( ) functions shall return 0 if they succeed.

14593 CX Otherwise, they shall return –1 and set *errno* to indicate the error.

14594 **ERRORS**

14595 CX The *fseek*( ) and *fseeko*( ) functions shall fail if, either the *stream* is unbuffered or the *stream*'s
14596 buffer needed to be flushed, and the call to *fseek*( ) or *fseeko*( ) causes an underlying *lseek*( ) or
14597 *write*( ) to be invoked, and:


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale
```

14598 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be
14599 delayed in the write operation.

14600 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the
14601 stream's buffer needed to be flushed and the file is not open.

14602 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

14603 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

14604 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
14605 offset maximum associated with the corresponding stream.

14606 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
14607 was transferred.

14608 CX [EINVAL] The *whence* argument is invalid. The resulting file-position indicator would be
14609 set to a negative value.

14610 CX [EIO] A physical I/O error has occurred, or the process is a member of a
14611 background process group attempting to perform a *write*() to its controlling
14612 terminal, TOSTOP is set, the process is neither ignoring nor blocking
14613 SIGTTOU, and the process group of the process is orphaned. This error may
14614 also be returned under implementation-defined conditions.

14615 CX [ENOSPC] There was no free space remaining on the device containing the file.

14616 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
14617 capabilities of the device.

14618 CX [EOVERFLOW] For *fseek*(), the resulting file offset would be a value which cannot be
14619 represented correctly in an object of type **long**.

14620 CX [EOVERFLOW] For *fseeko*(), the resulting file offset would be a value which cannot be
14621 represented correctly in an object of type **off_t**.

14622 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading
14623 by any process; a SIGPIPE signal shall also be sent to the thread.

14624 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14625 **EXAMPLES**

14626 None.

14627 **APPLICATION USAGE**

14628 None.

14629 **RATIONALE**

14630 None.

14631 **FUTURE DIRECTIONS**

14632 None.

14633 **SEE ALSO**

14634 *fopen*(), *fsetpos*(), *ftell*(), *getrlimit*(), *lseek*(), *rewind*(), *ulimit*(), *ungetc*(), *write*(), the Base
14635 Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

14636 **CHANGE HISTORY**

14637 First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *fseeko*( ) function is added.
- The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

The following change is incorporated for alignment with the FIPS requirements:

- The [EINTR] error is no longer an indication that the implementation does not report partial transfers.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The DESCRIPTION is updated to explicitly state that *fseek*( ) sets the file-position indicator, and then on error the error indicator is set and *fseek*( ) fails. This is for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14655 **NAME**

14656 fsetpos — set current file position

14657 **SYNOPSIS**

14658
```
#include <stdio.h>
```

14659
```
int fsetpos(FILE *stream, const fpos_t *pos);
```

14660 **DESCRIPTION**

14661 CX The functionality described on this reference page is aligned with the ISO C standard. Any
14662 conflict between the requirements described here and the ISO C standard is unintentional. This
14663 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14664 The *fsetpos*( ) function shall set the file position and state indicators for the stream pointed to by
14665 *stream* according to the value of the object pointed to by *pos*, which the application shall ensure
14666 is a value obtained from an earlier call to *fgetpos*( ) on the same stream. If a read or write error
14667 occurs, the error indicator for the stream shall be set and *fsetpos*( ) fails.

14668 A successful call to the *fsetpos*( ) function shall clear the end-of-file indicator for the stream and
14669 undo any effects of *ungetc*( ) on the same stream. After an *fsetpos*( ) call, the next operation on an
14670 update stream may be either input or output.

14671 CX The behavior of *fsetpos*( ) on devices which are incapable of seeking is implementation-defined.
14672 The value of the file offset associated with such a device is undefined.

14673 **RETURN VALUE**

14674 The *fsetpos*( ) function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and
14675 set *errno* to indicate the error.

14676 **ERRORS**

14677 CX The *fsetpos*( ) function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to
14678 be flushed, and the call to *fsetpos*( ) causes an underlying *lseek*( ) or *write*( ) to be invoked, and:

14679 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be
14680 delayed in the write operation.

14681 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the
14682 stream's buffer needed to be flushed and the file is not open.

14683 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

14684 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

14685 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
14686 offset maximum associated with the corresponding stream.

14687 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
14688 was transferred.

14689 CX [EIO] A physical I/O error has occurred, or the process is a member of a
14690 background process group attempting to perform a *write*( ) to its controlling
14691 terminal, TOSTOP is set, the process is neither ignoring nor blocking
14692 SIGTTOU, and the process group of the process is orphaned. This error may
14693 also be returned under implementation-defined conditions.

14694 CX [ENOSPC] There was no free space remaining on the device containing the file.

14695 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
14696 capabilities of the device.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14697 CX [EPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14698 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading
14699 by any process; a SIGPIPE signal shall also be sent to the thread.

14700 **EXAMPLES**

14701 None.

14702 **APPLICATION USAGE**

14703 None.

14704 **RATIONALE**

14705 None.

14706 **FUTURE DIRECTIONS**

14707 None.

14708 **SEE ALSO**

14709 *fopen*( ), *ftell*( ), *lseek*( ), *rewind*( ), *ungetc*( ), *write*( ), the Base Definitions volume of
14710 IEEE Std 1003.1-2001, **<stdio.h>**

14711 **CHANGE HISTORY**

14712 First released in Issue 4. Derived from the ISO C standard.

14713 **Issue 6**

14714 Extensions beyond the ISO C standard are marked.

14715 An additional [ESPIPE] error condition is added for sockets.

14716 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

14717 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or
14718 write error. This is for alignment with the ISO/IEC 9899: 1999 standard.

14719 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous
14720 [EINVAL] error case from the ERRORS section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

14721 **NAME**

14722 fstat — get file status

14723 **SYNOPSIS**

14724
```
#include <sys/stat.h>
```

14725
```
int fstat(int fildes, struct stat *buf);
```

14726 **DESCRIPTION**

14727 The *fstat*( ) function shall obtain information about an open file associated with the file
14728 descriptor *fildes*, and shall write it to the area pointed to by *buf*.

14729 SHM If *fildes* references a shared memory object, the implementation shall update in the **stat** structure
14730 pointed to by the *buf* argument only the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
14731 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
14732 valid. The implementation may update other fields and flags.

14733 TYM If *fildes* references a typed memory object, the implementation shall update in the **stat** structure
14734 pointed to by the *buf* argument only the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
14735 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
14736 valid. The implementation may update other fields and flags.

14737 The *buf* argument is a pointer to a **stat** structure, as defined in **<sys/stat.h>**, into which
14738 information is placed concerning the file.

14739 The structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*, *st_ctime*, and *st_mtime*
14740 shall have meaningful values for all other file types defined in this volume of
14741 IEEE Std 1003.1-2001. The value of the member *st_nlink* shall be set to the number of links to the
14742 file.

14743 An implementation that provides additional or alternative file access control mechanisms may,
14744 under implementation-defined conditions, cause *fstat*( ) to fail.

14745 The *fstat*( ) function shall update any time-related fields as described in the Base Definitions
14746 volume of IEEE Std 1003.1-2001, Section 4.7, File Times Update, before writing into the **stat**
14747 structure.

14748 **RETURN VALUE**

14749 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
14750 indicate the error.

14751 **ERRORS**

14752 The *fstat*( ) function shall fail if:

14753 [EBADF] The *fildes* argument is not a valid file descriptor.

14754 [EIO] An I/O error occurred while reading from the file system.

14755 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
14756 serial number cannot be represented correctly in the structure pointed to by
14757 *buf*.

14758 The *fstat*( ) function may fail if:

14759 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*
14760 argument.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Obtaining File Status Information

The following example shows how to obtain file status information for a file named **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file descriptor *fildes*.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct stat buffer;
int         status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = fstat(fildes, &buffer);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

lstat( ), *stat*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EIO] mandatory error condition is added.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [EOVERFLOW] optional error condition is added.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that shared memory object semantics apply to typed memory objects.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fstatvfs, statvfs — get file system information

SYNOPSIS

XSI

```
#include <sys/statvfs.h>

int fstatvfs(int fildes, struct statvfs *buf);
int statvfs(const char *restrict path, struct statvfs *restrict buf);
```

DESCRIPTION

The *fstatvfs*( ) function shall obtain information about the file system containing the file referenced by *fildes*.

The *statvfs*( ) function shall obtain information about the file system containing the file named by *path*.

For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read, write, or execute permission of the named file is not required.

The following flags can be returned in the *f_flag* member:

ST_RDONLY Read-only file system.

ST_NOSUID Setuid/setgid bits ignored by *exec*.

It is unspecified whether all members of the **statvfs** structure have meaningful values on all file systems.

RETURN VALUE

Upon successful completion, *statvfs*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *fstatvfs*( ) and *statvfs*( ) functions shall fail if:

[EIO] An I/O error occurred while reading the file system.

[EINTR] A signal was caught during execution of the function.

[EOVERFLOW] One of the values to be returned cannot be represented correctly in the structure pointed to by *buf*.

The *fstatvfs*( ) function shall fail if:

[EBADF] The *fildes* argument is not an open file descriptor.

The *statvfs*( ) function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix of *path* is not a directory.

--`,,,`--`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *statvfs*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

EXAMPLES

Obtaining File System Information Using fstatvfs( )

The following example shows how to obtain file system information for the file system upon which the file named **/home/cnd/mod1** resides, using the *fstatvfs*( ) function. The **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed to the *fstatvfs*( ) function.

```
#include <statvfs.h>
#include <fcntl.h>

struct statvfs buffer;
int            status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status  = fstatvfs(fildes, &buffer);
```

Obtaining File System Information Using statvfs( )

The following example shows how to obtain file system information for the file system upon which the file named **/home/cnd/mod1** resides, using the *statvfs*( ) function.

```
#include <statvfs.h>

struct statvfs buffer;
int            status;
...
status = statvfs("/home/cnd/mod1", &buffer);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *chown*( ), *creat*( ), *dup*( ), *exec*, *fcntl*( ), *link*( ), *mknod*( ), *open*( ), *pipe*( ), *read*( ), *time*( ), *unlink*( ), *utime*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/statvfs.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Large File Summit extensions are added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *statvfs*() prototype for alignment with the ISO/IEC 9899:1999 standard.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14887 **NAME**

14888 fsync — synchronize changes to a file

14889 **SYNOPSIS**

14890 FSC
```
#include <unistd.h>
```

14891
```
int fsync(int fildes);
```

14892

14893 **DESCRIPTION**

14894 The *fsync*( ) function shall request that all data for the open file descriptor named by *fildes* is to be
14895 transferred to the storage device associated with the file described by *fildes* in an
14896 implementation-defined manner. The *fsync*( ) function shall not return until the system has
14897 completed that action or until an error is detected.

14898 SIO If _POSIX_SYNCHRONIZED_IO is defined, the *fsync*( ) function shall force all currently queued
14899 I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O
14900 completion state. All I/O operations shall be completed as defined for synchronized I/O file
14901 integrity completion.

14902 **RETURN VALUE**

14903 Upon successful completion, *fsync*( ) shall return 0. Otherwise, –1 shall be returned and *errno* set
14904 to indicate the error. If the *fsync*( ) function fails, outstanding I/O operations are not guaranteed
14905 to have been completed.

14906 **ERRORS**

14907 The *fsync*( ) function shall fail if:

14908 [EBADF] The *fildes* argument is not a valid descriptor.

14909 [EINTR] The *fsync*( ) function was interrupted by a signal.

14910 [EINVAL] The *fildes* argument does not refer to a file on which this operation is possible.

14911 [EIO] An I/O error occurred while reading from or writing to the file system.

14912 In the event that any of the queued I/O operations fail, *fsync*( ) shall return the error conditions
14913 defined for *read*( ) and *write*( ).

14914 **EXAMPLES**

14915 None.

14916 **APPLICATION USAGE**

14917 The *fsync*( ) function should be used by programs which require modifications to a file to be
14918 completed before continuing; for example, a program which contains a simple transaction
14919 facility might use it to ensure that all modifications to a file or files caused by a transaction are
14920 recorded.

14921 **RATIONALE**

14922 The *fsync*( ) function is intended to force a physical write of data from the buffer cache, and to
14923 assure that after a system crash or other failure that all data up to the time of the *fsync*( ) call is
14924 recorded on the disk. Since the concepts of ‘‘buffer cache’’, ‘‘system crash’’, ‘‘physical write’’, and
14925 ‘‘non-volatile storage’’ are not defined here, the wording has to be more abstract.

14926 If _POSIX_SYNCHRONIZED_IO is not defined, the wording relies heavily on the conformance
14927 document to tell the user what can be expected from the system. It is explicitly intended that a
14928 null implementation is permitted. This could be valid in the case where the system cannot assure
14929 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the
14930 functionality is not required. In the middle ground between these extremes, *fsync*( ) might or
14931 might not actually cause data to be written where it is safe from a power failure. The

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

conformance document should identify at least that one configuration exists (and how to obtain that configuration) where this can be assured for at least some files that the user can select to use for critical data. It is not intended that an exhaustive list is required, but rather sufficient information is provided so that if critical data needs to be saved, the user can determine how the system is to be configured to allow the data to be written to non-volatile storage.

It is reasonable to assert that the key aspects of *fsync*( ) are unreasonable to test in a test suite. That does not make the function any less valuable, just more difficult to test. A formal conformance test should probably force a system crash (power shutdown) during the test for this condition, but it needs to be done in such a way that automated testing does not require this to be done except when a formal record of the results is being made. It would also not be unreasonable to omit testing for *fsync*( ), allowing it to be treated as a quality-of-implementation issue.

FUTURE DIRECTIONS

None.

SEE ALSO

sync( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

Aligned with *fsync*( ) in the POSIX Realtime Extension. Specifically, the DESCRIPTION and RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate that *fsync*( ) can return the error conditions defined for *read*( ) and *write*( ).

Issue 6

This function is marked as part of the File Synchronization option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] and [EIO] mandatory error conditions are added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14959 **NAME**

14960 ftell, ftello — return a file offset in a stream

14961 **SYNOPSIS**

14962 `#include <stdio.h>`

14963 `long ftell(FILE *stream);`

14964 CX `off_t ftello(FILE *stream);`

14965

14966 **DESCRIPTION**

14967 CX The functionality described on this reference page is aligned with the ISO C standard. Any
14968 conflict between the requirements described here and the ISO C standard is unintentional. This
14969 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14970 The *ftell*( ) function shall obtain the current value of the file-position indicator for the stream
14971 pointed to by *stream*.

14972 CX The *ftello*( ) function shall be equivalent to *ftell*( ), except that the return value is of type **off_t**.

14973 **RETURN VALUE**

14974 CX Upon successful completion, *ftell*( ) and *ftello*( ) shall return the current value of the file-position
14975 indicator for the stream measured in bytes from the beginning of the file.

14976 CX Otherwise, *ftell*( ) and *ftello*( ) shall return –1, cast to **long** and **off_t** respectively, and set *errno* to
14977 indicate the error.

14978 **ERRORS**

14979 CX The *ftell*( ) and *ftello*( ) functions shall fail if:

14980 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.

14981 CX [EOVERFLOW] For *ftell*( ), the current file offset cannot be represented correctly in an object of
14982 type **long**.

14983 CX [EOVERFLOW] For *ftello*( ), the current file offset cannot be represented correctly in an object
14984 of type **off_t**.

14985 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14986 The *ftell*( ) function may fail if:

14987 CX [ESPIPE] The file descriptor underlying *stream* is associated with a socket.

14988 **EXAMPLES**

14989 None.

14990 **APPLICATION USAGE**

14991 None.

14992 **RATIONALE**

14993 None.

14994 **FUTURE DIRECTIONS**

14995 None.

14996 **SEE ALSO**

14997 *fgetpos*( ), *fopen*( ), *fseek*( ), *lseek*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14998 **CHANGE HISTORY**

14999 First released in Issue 1. Derived from Issue 1 of the SVID.

15000 **Issue 5**

15001 Large File Summit extensions are added.

15002 **Issue 6**

15003 Extensions beyond the ISO C standard are marked.

15004 The following new requirements on POSIX implementations derive from alignment with the
15005 Single UNIX Specification:

15006 • The *ftello*( ) function is added.

15007 • The [EOVERFLOW] error conditions are added.

15008 An additional [ESPIPE] error condition is added for sockets.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15009 **NAME**

15010 ftime — get date and time (**LEGACY**)

15011 **SYNOPSIS**

15012 XSI
```
#include <sys/timeb.h>

int ftime(struct timeb *tp);
```
15013

15014

15015 **DESCRIPTION**

15016 The *ftime*( ) function shall set the *time* and *millitm* members of the **timeb** structure pointed to by
15017 *tp* to contain the seconds and milliseconds portions, respectively, of the current time in seconds
15018 since the Epoch. The contents of the *timezone* and *dstflag* members of *tp* after a call to *ftime*( ) are
15019 unspecified.

15020 The system clock need not have millisecond granularity. Depending on any granularity
15021 (particularly a granularity of one) renders code non-portable.

15022 **RETURN VALUE**

15023 Upon successful completion, the *ftime*( ) function shall return 0; otherwise, –1 shall be returned.

15024 **ERRORS**

15025 No errors are defined.

15026 **EXAMPLES**

15027 **Getting the Current Time and Date**

15028 The following example shows how to get the current system time values using the *ftime*( )
15029 function. The **timeb** structure pointed to by *tp* is filled with the current system time values for
15030 *time* and *millitm*.

```
15031 #include <sys/timeb.h>

15032 struct timeb tp;
15033 int          status;
15034 ...
15035 status = ftime(&tp);
```

15036 **APPLICATION USAGE**

15037 For applications portability, the *time*( ) function should be used to determine the current time
15038 instead of *ftime*( ). Realtime applications should use *clock_gettime*( ) to determine the current
15039 time instead of *ftime*( ).

15040 **RATIONALE**

15041 None.

15042 **FUTURE DIRECTIONS**

15043 This function may be withdrawn in a future version.

15044 **SEE ALSO**

15045 *clock_getres*( ), *ctime*( ), *gettimeofday*( ), *time*( ), the Base Definitions volume of
15046 IEEE Std 1003.1-2001, **<sys/timeb.h>**

15047 **CHANGE HISTORY**

15048 First released in Issue 4, Version 2.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15049 **Issue 5**

15050 Moved from X/OPEN UNIX extension to BASE.

15051 Normative text previously in the APPLICATION USAGE section is moved to the
15052 DESCRIPTION.

15053 **Issue 6**

15054 This function is marked LEGACY.

15055 The DESCRIPTION is updated to refer to ‘‘seconds since the Epoch’’ rather than ‘‘seconds since
15056 00:00:00 UTC (Coordinated Universal Time), January 1 1970’’ for consistency with other *time*
15057 functions.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15058 **NAME**

15059 ftok — generate an IPC key

15060 **SYNOPSIS**

15061 XSI
```
#include <sys/ipc.h>

key_t ftok(const char *path, int id);
```
15062

15063

15064 **DESCRIPTION**

15065 The *ftok*( ) function shall return a key based on *path* and *id* that is usable in subsequent calls to
15066 *msgget*( ), *semget*( ), and *shmget*( ). The application shall ensure that the *path* argument is the
15067 pathname of an existing file that the process is able to *stat*( ).

15068 The *ftok*( ) function shall return the same key value for all paths that name the same file, when
15069 called with the same *id* value, and return different key values when called with different *id*
15070 values or with paths that name different files existing on the same file system at the same time. It
15071 is unspecified whether *ftok*( ) shall return the same key value when called again after the file
15072 named by *path* is removed and recreated with the same name.

15073 Only the low-order 8-bits of *id* are significant. The behavior of *ftok*( ) is unspecified if these bits
15074 are 0.

15075 **RETURN VALUE**

15076 Upon successful completion, *ftok*( ) shall return a key. Otherwise, *ftok*( ) shall return (**key_t**)–1
15077 and set *errno* to indicate the error.

15078 **ERRORS**

15079 The *ftok*( ) function shall fail if:

15080 [EACCES] Search permission is denied for a component of the path prefix.

15081 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
15082 argument.

15083 [ENAMETOOLONG]
15084 The length of the *path* argument exceeds {PATH_MAX} or a pathname
15085 component is longer than {NAME_MAX}.

15086 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

15087 [ENOTDIR] A component of the path prefix is not a directory.

15088 The *ftok*( ) function may fail if:

15089 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
15090 resolution of the *path* argument.

15091 [ENAMETOOLONG]
15092 Pathname resolution of a symbolic link produced an intermediate result
15093 whose length exceeds {PATH_MAX}.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Getting an IPC Key

The following example gets a unique key that can be used by the IPC functions *semget*(), *msgget*(), and *shmget*(). The key returned by *ftok*() for this example is based on the ID value *S* and the pathname **/tmp**.

```
#include <sys/ipc.h>
...
key_t key;
char *path = "/tmp";
int id = 'S';

key = ftok(path, id);
```

Saving an IPC Key

The following example gets a unique key based on the pathname **/tmp** and the ID value *a*. It also assigns the value of the resulting key to the *semkey* variable so that it will be available to a later call to *semget*(), *msgget*(), or *shmget*().

```
#include <sys/ipc.h>
...
key_t semkey;

if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
    perror("IPC error: ftok"); exit(1);
}
```

APPLICATION USAGE

For maximum portability, *id* should be a single-byte character.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

msgget(), *semget*(), *shmget*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/ipc.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15131 **NAME**

15132 ftruncate — truncate a file to a specified length

15133 **SYNOPSIS**

15134 `#include <unistd.h>`

15135 `int ftruncate(int fildes, off_t length);`

15136 **DESCRIPTION**

15137 If *fildes* is not a valid file descriptor open for writing, the *ftruncate*( ) function shall fail.

15138 If *fildes* refers to a regular file, the *ftruncate*( ) function shall cause the size of the file to be
15139 truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no
15140 longer be available to reads on the file. If the file previously was smaller than this size,
15141 XSI *ftruncate*( ) shall either increase the size of the file or fail. XSI-conformant systems shall increase
15142 the size of the file. If the file size is increased, the extended area shall appear as if it were zero-
15143 filled. The value of the seek pointer shall not be modified by a call to *ftruncate*( ).

15144 Upon successful completion, if *fildes* refers to a regular file, the *ftruncate*( ) function shall mark
15145 for update the *st_ctime* and *st_mtime* fields of the file and the S_ISUID and S_ISGID bits of the file
15146 mode may be cleared. If the *ftruncate*( ) function is unsuccessful, the file is unaffected.

15147 XSI If the request would cause the file size to exceed the soft file size limit for the process, the
15148 request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

15149 If *fildes* refers to a directory, *ftruncate*( ) shall fail.

15150 If *fildes* refers to any other file type, except a shared memory object, the result is unspecified.

15151 SHM If *fildes* refers to a shared memory object, *ftruncate*( ) shall set the size of the shared memory
15152 object to *length*.

15153 MF|SHM If the effect of *ftruncate*( ) is to decrease the size of a shared memory object or memory mapped
15154 file and whole pages beyond the new end were previously mapped, then the whole pages
15155 beyond the new end shall be discarded.

15156 MPR If the Memory Protection option is supported, references to discarded pages shall result in the
15157 generation of a SIGBUS signal; otherwise, the result of such references is undefined.

15158 MF|SHM If the effect of *ftruncate*( ) is to increase the size of a shared memory object, it is unspecified
15159 whether the contents of any mapped pages between the old end-of-file and the new are flushed
15160 to the underlying object.

15161 **RETURN VALUE**

15162 Upon successful completion, *ftruncate*( ) shall return 0; otherwise, –1 shall be returned and *errno*
15163 set to indicate the error.

15164 **ERRORS**

15165 The *ftruncate*( ) function shall fail if:

15166 [EINTR] A signal was caught during execution.

15167 [EINVAL] The *length* argument was less than 0.

15168 [EFBIG] or [EINVAL]
15169 The *length* argument was greater than the maximum file size.

15170 XSI [EFBIG] The file is a regular file and *length* is greater than the offset maximum
15171 established in the open file description associated with *fildes*.

15172 [EIO] An I/O error occurred while reading from or writing to a file system.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15173 [EBADF] or [EINVAL]
15174 The *fildes* argument is not a file descriptor open for writing.

15175 [EINVAL] The *fildes* argument references a file that was opened without write
15176 permission.

15177 [EROFS] The named file resides on a read-only file system.

15178 **EXAMPLES**

15179 None.

15180 **APPLICATION USAGE**

15181 None.

15182 **RATIONALE**

15183 The *ftruncate*( ) function is part of IEEE Std 1003.1-2001 as it was deemed to be more useful than
15184 *truncate*( ). The *truncate*( ) function is provided as an XSI extension.

15185 **FUTURE DIRECTIONS**

15186 None.

15187 **SEE ALSO**

15188 *open*( ), *truncate*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

15189 **CHANGE HISTORY**

15190 First released in Issue 4, Version 2.

15191 **Issue 5**

15192 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate*( ) in the POSIX
15193 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is
15194 added to the list of mandatory errors that can be returned by *ftruncate*( ).

15195 Large File Summit extensions are added.

15196 **Issue 6**

15197 The *truncate*( ) function is split out into a separate reference page.

15198 The following new requirements on POSIX implementations derive from alignment with the
15199 Single UNIX Specification:

15200 • The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a
15201 regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.

15202 The following changes were made to align with the IEEE P1003.1a draft standard:

15203 • The DESCRIPTION text is updated.

15204 XSI-conformant systems are required to increase the size of the file if the file was previously
15205 smaller than the size requested.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15206 **NAME**

15207 ftrylockfile — stdio locking functions

15208 **SYNOPSIS**

15209 TSF
```
#include <stdio.h>
```

15210
```
int ftrylockfile(FILE *file);
```

15211

15212 **DESCRIPTION**

15213 Refer to *flockfile*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ftw — traverse (walk) a file tree

SYNOPSIS

XSI
```
#include <ftw.h>

int ftw(const char *path, int (*fn)(const char *,
    const struct stat *ptr, int flag), int ndirs);
```

DESCRIPTION

The *ftw*( ) function shall recursively descend the directory hierarchy rooted in *path*. For each object in the hierarchy, *ftw*( ) shall call the function pointed to by *fn*, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a **stat** structure containing information about the object, and an integer. Possible values of the integer, defined in the **<ftw.h>** header, are:

FTW_D For a directory.

FTW_DNR For a directory that cannot be read.

FTW_F For a file.

FTW_SL For a symbolic link (but see also FTW_NS below).

FTW_NS For an object other than a symbolic link on which *stat*( ) could not successfully be executed. If the object is a symbolic link and *stat*( ) failed, it is unspecified whether *ftw*( ) passes FTW_SL or FTW_NS to the user-supplied function.

If the integer is FTW_DNR, descendants of that directory shall not be processed. If the integer is FTW_NS, the **stat** structure contains undefined values. An example of an object that would cause FTW_NS to be passed to the function pointed to by *fn* would be a file in a directory with read but without execute (search) permission.

The *ftw*( ) function shall visit a directory before visiting any of its descendants.

The *ftw*( ) function shall use at most one file descriptor for each level in the tree.

The argument *ndirs* should be in the range [1,{OPEN_MAX}].

The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a non-zero value, or some error, other than [EACCES], is detected within *ftw*( ).

The *ndirs* argument shall specify the maximum number of directory streams or file descriptors or both available for use by *ftw*( ) while traversing the tree. When *ftw*( ) returns it shall close any directory streams and file descriptors it uses not counting any opened by the application-supplied *fn* function.

The results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

The *ftw*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

If the tree is exhausted, *ftw*( ) shall return 0. If the function pointed to by *fn* returns a non-zero value, *ftw*( ) shall stop its tree traversal and return whatever value was returned by the function pointed to by *fn*. If *ftw*( ) detects an error, it shall return –1 and set *errno* to indicate the error.

If *ftw*( ) encounters an error other than [EACCES] (see FTW_DNR and FTW_NS above), it shall return –1 and set *errno* to indicate the error. The external variable *errno* may contain any error

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

value that is possible when a directory is opened or when one of the *stat* functions is executed on a directory or file.

ERRORS

The *ftw*( ) function shall fail if:

[EACCES] Search permission is denied for any component of *path* or read permission is denied for *path*.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of *path* is not a directory.

[EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current programming environment for one or more files found in the file hierarchy.

The *ftw*( ) function may fail if:

[EINVAL] The value of the *ndirs* argument is invalid.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set accordingly.

EXAMPLES

Walking a Directory Structure

The following example walks the current directory structure, calling the *fn* function for every directory entry, using at most 10 file descriptors:

```
#include <ftw.h>
...
if (ftw(".", fn, 10) != 0) {
    perror("ftw"); exit(2);
}
```

APPLICATION USAGE

The *ftw*( ) function may allocate dynamic storage during its operation. If *ftw*( ) is forcibly terminated, such as by *longjmp*( ) or *siglongjmp*( ) being executed by the function pointed to by *fn* or an interrupt routine, *ftw*( ) does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next invocation.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15297 **RATIONALE**

15298 None.

15299 **FUTURE DIRECTIONS**

15300 None.

15301 **SEE ALSO**

15302 *longjmp*( ), *lstat*( ), *malloc*( ), *nftw*( ), *opendir*( ), *siglongjmp*( ), *stat*( ), the Base Definitions volume of
15303 IEEE Std 1003.1-2001, **<ftw.h>**, **<sys/stat.h>**

15304 **CHANGE HISTORY**

15305 First released in Issue 1. Derived from Issue 1 of the SVID.

15306 **Issue 5**

15307 UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections are changed to EX.

15308 **Issue 6**

15309 The ERRORS section is updated as follows:

15310 • The wording of the mandatory [ELOOP] error condition is updated.

15311 • A second optional [ELOOP] error condition is added.

15312 • The [EOVERFLOW] mandatory error condition is added.

15313 Text is added to the DESCRIPTION to say that the *ftw*( ) function need not be reentrant and that
15314 the results are unspecified if the application-supplied *fn* function does not preserve the current
15315 working directory.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15316 **NAME**

15317 funlockfile — stdio locking functions

15318 **SYNOPSIS**

15319 TSF
```
#include <stdio.h>
```

15320
```
void funlockfile(FILE *file);
```

15321

15322 **DESCRIPTION**

15323 Refer to *flockfile*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15324 **NAME**

15325 fwide — set stream orientation

15326 **SYNOPSIS**

15327 #include <stdio.h>
15328 #include <wchar.h>

15329 int fwide(FILE *stream, int mode);

15330 **DESCRIPTION**

15331 CX The functionality described on this reference page is aligned with the ISO C standard. Any
15332 conflict between the requirements described here and the ISO C standard is unintentional. This
15333 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15334 The *fwide*( ) function shall determine the orientation of the stream pointed to by *stream*. If *mode* is
15335 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less
15336 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero
15337 and the function does not alter the orientation of the stream.

15338 If the orientation of the stream has already been determined, *fwide*( ) shall not change it.

15339 CX Since no return value is reserved to indicate an error, an application wishing to check for error
15340 situations should set *errno* to 0, then call *fwide*( ), then check *errno*, and if it is non-zero, assume
15341 an error has occurred.

15342 **RETURN VALUE**

15343 The *fwide*( ) function shall return a value greater than zero if, after the call, the stream has wide-
15344 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no
15345 orientation.

15346 **ERRORS**

15347 The *fwide*( ) function may fail if:

15348 CX [EBADF] The *stream* argument is not a valid stream.

15349 **EXAMPLES**

15350 None.

15351 **APPLICATION USAGE**

15352 A call to *fwide*( ) with *mode* set to zero can be used to determine the current orientation of a
15353 stream.

15354 **RATIONALE**

15355 None.

15356 **FUTURE DIRECTIONS**

15357 None.

15358 **SEE ALSO**

15359 The Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

15360 **CHANGE HISTORY**

15361 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
15362 (E).

15363 **Issue 6**

15364 Extensions beyond the ISO C standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15365 **NAME**

15366 fwprintf, swprintf, wprintf — print formatted wide-character output

15367 **SYNOPSIS**

15368 ```
#include <stdio.h>
```
15369 ```
#include <wchar.h>
```

15370 ```
int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
```
15371 ```
int swprintf(wchar_t *restrict ws, size_t n,
```
15372 ```
    const wchar_t *restrict format, ...);
```
15373 ```
int wprintf(const wchar_t *restrict format, ...);
```

15374 **DESCRIPTION**

15375 CX The functionality described on this reference page is aligned with the ISO C standard. Any
15376 conflict between the requirements described here and the ISO C standard is unintentional. This
15377 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15378 The *fwprintf*( ) function shall place output on the named output *stream*. The *wprintf*( ) function
15379 shall place output on the standard output stream *stdout*. The *swprintf*( ) function shall place
15380 output followed by the null wide character in consecutive wide characters starting at **ws*; no
15381 more than *n* wide characters shall be written, including a terminating null wide character, which
15382 is always added (unless *n* is zero).

15383 Each of these functions shall convert, format, and print its arguments under control of the *format*
15384 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-*
15385 *characters*, which are simply copied to the output stream, and *conversion specifications*, each of
15386 which results in the fetching of zero or more arguments. The results are undefined if there are
15387 insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the
15388 excess arguments are evaluated but are otherwise ignored.

15389 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
15390 to the next unused argument. In this case, the conversion specifier wide character `%` (see below)
15391 is replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}],
15392 giving the position of the argument in the argument list. This feature provides for the definition
15393 of *format* wide-character strings that select arguments in an order appropriate to specific
15394 languages (see the EXAMPLES section).

15395 The *format* can contain either numbered argument specifications (that is, `"%`*n*`$"` and `"*`*m*`$"`), or
15396 unnumbered argument conversion specifications (that is, `%` and `*`), but not both. The only
15397 exception to this is that `%%` can be mixed with the `"%`*n*`$"` form. The results of mixing numbered
15398 and unnumbered argument specifications in a *format* wide-character string are undefined. When
15399 numbered argument specifications are used, specifying the *N*th argument requires that all the
15400 leading arguments, from the first to the (*N*–1)th, are specified in the *format* wide-character string.

15401 In *format* wide-character strings containing the `"%`*n*`$"` form of conversion specification,
15402 numbered arguments in the argument list can be referenced from the *format* wide-character
15403 string as many times as required.

15404 In *format* wide-character strings containing the `%` form of conversion specification, each
15405 argument in the argument list shall be used exactly once.

15406 CX All forms of the *fwprintf*( ) function allow for the insertion of a locale-dependent radix character
15407 in the output string, output as a wide-character value. The radix character is defined in the
15408 program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix
15409 character is not defined, the radix character shall default to a period (`'.'`).

15410 XSI Each conversion specification is introduced by the `'%'` wide character or by the wide-character
15411 sequence `"%`*n*`$"`, after which the following appear in sequence:

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- An optional minimum *field width*. If the converted value has fewer wide characters than the field width, it shall be padded with spaces by default on the left; it shall be padded on the right, if the left-adjustment flag ('−'), described below, is given to the field width. The field width takes the form of an asterisk ('*'), described below, or a decimal integer.
- An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the number of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g and G conversion specifiers; or the maximum number of wide characters to be printed from a string in the s conversion specifiers. The precision takes the form of a period ('.') followed either by an asterisk ('*'), described below, or an optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion wide character, the behavior is undefined.
- An optional length modifier that specifies the size of the argument.
- A *conversion specifier* wide character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an asterisk ('*'). In this case an argument of type **int** supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a '−' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In *format* wide-character strings containing the "%*n*$" form of a conversion specification, a field width or precision may be indicated by the sequence "**m*$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an integer argument containing the field width or precision, for example:

```
wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The flag wide characters and their meanings are:

' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping wide characters. For other conversions, the behavior is undefined. The numeric grouping wide character is used.

− The result of the conversion shall be left-justified within the field. The conversion shall be right-justified if this flag is not specified.

+ The result of a signed conversion shall always begin with a sign ('+' or '−'). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.

<space> If the first wide character of a signed conversion is not a sign, or if a signed conversion results in no wide characters, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

Specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be 0. For x or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversion specifiers, trailing zeros shall *not* be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15457 `0` For `d`, `i`, `o`, `u`, `x`, `X`, `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, leading zeros
15458 (following any indication of sign or base) are used to pad to the field width; no space
15459 padding is performed. If the `'0'` and `'-'` flags both appear, the `'0'` flag shall be
15460 ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversion specifiers, if a precision is specified, the `'0'`
15461 flag shall be ignored. If the `'0'` and `'''` flags both appear, the grouping wide
15462 characters are inserted before zero padding. For other conversions, the behavior is
15463 undefined.

15464 The length modifiers and their meanings are:

15465 `hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **signed char**
15466 or **unsigned char** argument (the argument will have been promoted according to the
15467 integer promotions, but its value shall be converted to **signed char** or **unsigned char**
15468 before printing); or that a following `n` conversion specifier applies to a pointer to a
15469 **signed char** argument.

15470 `h` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **short** or
15471 **unsigned short** argument (the argument will have been promoted according to the
15472 integer promotions, but its value shall be converted to **short** or **unsigned short** before
15473 printing); or that a following `n` conversion specifier applies to a pointer to a **short**
15474 argument.

15475 `l` (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long** or
15476 **unsigned long** argument; that a following `n` conversion specifier applies to a pointer to
15477 a **long** argument; that a following `c` conversion specifier applies to a **wint_t** argument;
15478 that a following `s` conversion specifier applies to a pointer to a **wchar_t** argument; or
15479 has no effect on a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier.

15480 `ll` (ell-ell)
15481 Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long long** or
15482 **unsigned long long** argument; or that a following `n` conversion specifier applies to a
15483 pointer to a **long long** argument.

15484 `j` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to an **intmax_t**
15485 or **uintmax_t** argument; or that a following `n` conversion specifier applies to a pointer
15486 to an **intmax_t** argument.

15487 `z` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **size_t** or the
15488 corresponding signed integer type argument; or that a following `n` conversion specifier
15489 applies to a pointer to a signed integer type corresponding to a **size_t** argument.

15490 `t` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **ptrdiff_t** or
15491 the corresponding **unsigned** type argument; or that a following `n` conversion specifier
15492 applies to a pointer to a **ptrdiff_t** argument.

15493 `L` Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to a **long**
15494 **double** argument.

15495 If a length modifier appears with any conversion specifier other than as specified above, the
15496 behavior is undefined.

15497 The conversion specifiers and their meanings are:

15498 `d`, `i` The **int** argument shall be converted to a signed decimal in the style `"[-]`*`dddd`*`"`. The
15499 precision specifies the minimum number of digits to appear; if the value being
15500 converted can be represented in fewer digits, it shall be expanded with leading zeros.
15501 The default precision shall be 1. The result of converting zero with an explicit precision
15502 of zero shall be no wide characters.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`o` The **unsigned** argument shall be converted to unsigned octal format in the style `"dddd"`. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

`u` The **unsigned** argument shall be converted to unsigned decimal format in the style `"dddd"`. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

`x` The **unsigned** argument shall be converted to unsigned hexadecimal format in the style `"dddd"`; the letters `"abcdef"` are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

`X` Equivalent to the `x` conversion specifier, except that letters `"ABCDEF"` are used instead of `"abcdef"`.

`f, F` The **double** argument shall be converted to decimal notation in the style `"[–]`*`ddd.ddd`*`"`, where the number of digits after the radix character shall be equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no `'#'` flag is present, no radix character shall appear. If a radix character appears, at least one digit shall appear before it. The value shall be rounded in an implementation-defined manner to the appropriate number of digits.

A **double** argument representing an infinity shall be converted in one of the styles `"[–]inf"` or `"[–]infinity"`; which style is implementation-defined. A **double** argument representing a NaN shall be converted in one of the styles `"[–]nan"` or `"[–]nan(`*`n-char-sequence`*`)"`; which style, and the meaning of any *n-char-sequence*, is implementation-defined. The `F` conversion specifier produces `"INF"`, `"INFINITY"`, or `"NAN"` instead of `"inf"`, `"infinity"`, or `"nan"`, respectively.

`e, E` The **double** argument shall be converted in the style `"[–]`*`d.ddd`*`e±dd"`, where there shall be one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it shall be equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no `'#'` flag is present, no radix character shall appear. The value shall be rounded in an implementation-defined manner to the appropriate number of digits. The `E` conversion wide character shall produce a number with `'E'` instead of `'e'` introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an `f` or `F` conversion specifier.

`g, G` The **double** argument shall be converted in the style `f` or `e` (or in the style `F` or `E` in the case of a `G` conversion specifier), with the precision specifying the number of significant digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on the value converted; style `e` (or `E`) shall be used only if the exponent resulting from such a conversion is less than –4 or greater than or equal to the precision. Trailing zeros shall be removed from the fractional portion of the result; a radix character shall appear only if it is followed by a digit.

A **double** argument representing an infinity or NaN shall be converted in the style of an `f` or `F` conversion specifier.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`a,A` A **double** argument representing a floating-point number shall be converted in the style `"[-]0xh.hhhhp±d"`, where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type **double**, except that trailing zeros may be omitted; if the precision is zero and the `'#'` flag is not specified, no decimal-point wide character shall appear. The letters `"abcdef"` are used for `a` conversion and the letters `"ABCDEF"` for `A` conversion. The `A` conversion specifier produces a number with `'X'` and `'P'` instead of `'x'` and `'p'`. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an `f` or `F` conversion specifier.

`c` If no `l` (ell) qualifier is present, the **int** argument shall be converted to a wide character as if by calling the *btowc*( ) function and the resulting wide character shall be written. Otherwise, the **wint_t** argument shall be converted to **wchar_t**, and written.

`s` If no `l` (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the *mbrtowc*( ) function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.

If an `l` (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type **wchar_t**. Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.

`p` The application shall ensure that the argument is a pointer to **void**. The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.

`n` The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the *fwprintf*( ) functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.

XSI `C` Equivalent to `lc`.

XSI `S` Equivalent to `ls`.

`%` Output a `'%'` wide character; no argument shall be converted. The entire conversion specification shall be `%%`.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If a conversion specification does not match one of the above forms, the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by *fwprintf*( ) and *wprintf*( ) shall be printed as if *fputwc*( ) had been called.

For `a` and `A` conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

For `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L <= D <= U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.

CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the call to a successful execution of *fwprintf*( ) or *wprintf*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream, or a call to *exit*( ) or *abort*( ).

RETURN VALUE

Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide character in the case of *swprintf*( ), or a negative CX value if an output error was encountered, and set *errno* to indicate the error.

If *n* or more wide characters were requested to be written, *swprintf*( ) shall return a negative CX value, and set *errno* to indicate the error.

ERRORS

For the conditions under which *fwprintf*( ) and *wprintf*( ) fail and may fail, refer to *fputwc*( ).

In addition, all forms of *fwprintf*( ) may fail if:

XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been detected.

XSI [EINVAL] There are insufficient arguments.

In addition, *wprintf*( ) and *fwprintf*( ) may fail if:

XSI [ENOMEM] Insufficient storage space is available.

EXAMPLES

To print the language-independent date and time format, the following statement could be used:

```
wprintf(format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the wide-character string:

```
L"%s, %s %d, %d:%.2d\n"
```

producing the message:

```
Sunday, July 3, 10:02
```

whereas for German usage, *format* could be a pointer to the wide-character string:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

producing the message:

```
Sonntag, 3. Juli, 10:02
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

btowc( ), *fputwc*( ), *fwscanf*( ), *mbrtowc*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing the case if *n* or more wide characters are requested to be written using *swprintf*( ).

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The prototypes for *fwprintf*( ), *swprintf*( ), and *wprintf*( ) are updated.
- The DESCRIPTION is updated.
- The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
- The `a`, `A`, and `F` conversion characters are added.
- XSI shading is removed from the description of character string representations of infinity and NaN floating-point values.

The DESCRIPTION is updated to use the terms ''conversion specifier'' and ''conversion specification'' consistently.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fwrite — binary output

SYNOPSIS

```
#include <stdio.h>

size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
    FILE *restrict stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fwrite*( ) function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fputc*( ) function, taking the values (in order) from an array of **unsigned char** exactly overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by the number of bytes successfully written. If an error occurs, the resulting value of the file-position indicator for the stream is unspecified.

CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful execution of *fwrite*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream, or a call to *exit*( ) or *abort*( ).

RETURN VALUE

The *fwrite*( ) function shall return the number of elements successfully written, which may be less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite*( ) shall return 0 and the state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for the stream shall be set, CX and *errno* shall be set to indicate the error.

ERRORS

Refer to *fputc*( ).

EXAMPLES

None.

APPLICATION USAGE

Because of possible differences in element length and byte ordering, files written using *fwrite*( ) are application-dependent, and possibly cannot be read using *fread*( ) by a different application or by the same application on a different processor.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ferror( ), *fopen*( ), *printf*( ), *putc*( ), *puts*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following changes are made for alignment with the ISO / IEC 9899: 1999 standard:

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The *fwrite*( ) prototype is updated.
- The DESCRIPTION is updated to clarify how the data is written out using *fputc*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

fwscanf, swscanf, wscanf — convert formatted wide-character input

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwscanf(FILE *restrict stream, const wchar_t *restrict format, ... );
int swscanf(const wchar_t *restrict ws,
    const wchar_t *restrict format, ... );
int wscanf(const wchar_t *restrict format, ... );
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *fwscanf*( ) function shall read from the named input *stream*. The *wscanf*( ) function shall read from the standard input stream *stdin*. The *swscanf*( ) function shall read from the wide-character string *ws*. Each function reads wide characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control wide-character string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the *format* is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier wide character `%` (see below) is replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that select arguments in an order appropriate to specific languages. In *format* wide-character strings containing the `"%n$"` form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the *format* wide-character string more than once.

The *format* can contain either form of a conversion specification—that is, `%` or `"%n$"`— but the two forms cannot normally be mixed within a single *format* wide-character string. The only exception to this is that `%%` or `%*` can be mixed with the `"%n$"` form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*–1)th, are pointers.

CX The *fwscanf*( ) function in all its forms allows for detection of a language-dependent radix character in the input string, encoded as a wide-character value. The radix character is defined in the program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period (`'.'`).

The *format* is a wide-character string composed of zero or more directives. Each directive is composed of one of the following: one or more white-space wide characters (<space>s, <tab>s, <newline>s, <vertical-tab>s, or <form-feed>s); an ordinary wide character (neither `'%'` nor a white-space character); or a conversion specification. Each conversion specification is introduced
XSI by a `'%'` or the sequence `"%n$"` after which the following appear in sequence:

- An optional assignment-suppressing character `'*'`.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional length modifier that specifies the size of the receiving object.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15759 • A conversion specifier wide character that specifies the type of conversion to be applied. The
15760 valid conversion specifiers are described below.

15761 The *fwscanf*( ) functions shall execute each directive of the format in turn. If a directive fails, as
15762 detailed below, the function shall return. Failures are described as input failures (due to the
15763 unavailability of input bytes) or matching failures (due to inappropriate input).

15764 A directive composed of one or more white-space wide characters is executed by reading input
15765 until no more valid input can be read, or up to the first wide character which is not a white-
15766 space wide character, which remains unread.

15767 A directive that is an ordinary wide character shall be executed as follows. The next wide
15768 character is read from the input and compared with the wide character that comprises the
15769 directive; if the comparison shows that they are not equivalent, the directive shall fail, and the
15770 differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding
15771 error, or a read error prevents a wide character from being read, the directive shall fail.

15772 A directive that is a conversion specification defines a set of matching input sequences, as
15773 described below for each conversion wide character. A conversion specification is executed in
15774 the following steps.

15775 Input white-space wide characters (as specified by *iswspace*( )) shall be skipped, unless the
15776 conversion specification includes a `[`, `c`, or `n` conversion specifier.

15777 An item shall be read from the input, unless the conversion specification includes an `n`
15778 conversion specifier wide character. An input item is defined as the longest sequence of input
15779 wide characters, not exceeding any specified field width, which is an initial subsequence of a
15780 matching sequence. The first wide character, if any, after the input item shall remain unread. If
15781 the length of the input item is zero, the execution of the conversion specification shall fail; this
15782 condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented
15783 input from the stream, in which case it is an input failure.

15784 Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion
15785 specification, the count of input wide characters) shall be converted to a type appropriate to the
15786 conversion wide character. If the input item is not a matching sequence, the execution of the
15787 conversion specification shall fail; this condition is a matching failure. Unless assignment
15788 suppression was indicated by a `'*'`, the result of the conversion shall be placed in the object
15789 pointed to by the first argument following the *format* argument that has not already received a
15790 XSI conversion result if the conversion specification is introduced by `%`, or in the *n*th argument if
15791 introduced by the wide-character sequence `"%`*n*`$"`. If this object does not have an appropriate
15792 type, or if the result of the conversion cannot be represented in the space provided, the behavior
15793 is undefined.

15794 The length modifiers and their meanings are:

15795 `hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
15796 argument with type pointer to **signed char** or **unsigned char**.

15797 `h` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
15798 argument with type pointer to **short** or **unsigned short**.

15799 `l` (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
15800 argument with type pointer to **long** or **unsigned long**; that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`,
15801 or `G` conversion specifier applies to an argument with type pointer to **double**; or that a
15802 following `c`, `s`, or `[` conversion specifier applies to an argument with type pointer to
15803 **wchar_t**.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15804 `ll` (ell-ell)
15805 Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
15806 argument with type pointer to **long long** or **unsigned long long**.

15807 `j` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
15808 argument with type pointer to **intmax_t** or **uintmax_t**.

15809 `z` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
15810 argument with type pointer to **size_t** or the corresponding signed integer type.

15811 `t` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an
15812 argument with type pointer to **ptrdiff_t** or the corresponding **unsigned** type.

15813 `L` Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to an
15814 argument with type pointer to **long double**.

15815 If a length modifier appears with any conversion specifier other than as specified above, the
15816 behavior is undefined.

15817 The following conversion specifier wide characters are valid:

15818 `d` Matches an optionally signed decimal integer, whose format is the same as expected for
15819 the subject sequence of *wcstol*( ) with the value 10 for the *base* argument. In the absence
15820 of a size modifier, the application shall ensure that the corresponding argument is a
15821 pointer to **int**.

15822 `i` Matches an optionally signed integer, whose format is the same as expected for the
15823 subject sequence of *wcstol*( ) with 0 for the *base* argument. In the absence of a size
15824 modifier, the application shall ensure that the corresponding argument is a pointer to
15825 **int**.

15826 `o` Matches an optionally signed octal integer, whose format is the same as expected for
15827 the subject sequence of *wcstoul*( ) with the value 8 for the *base* argument. In the absence
15828 of a size modifier, the application shall ensure that the corresponding argument is a
15829 pointer to **unsigned**.

15830 `u` Matches an optionally signed decimal integer, whose format is the same as expected for
15831 the subject sequence of *wcstoul*( ) with the value 10 for the *base* argument. In the absence
15832 of a size modifier, the application shall ensure that the corresponding argument is a
15833 pointer to **unsigned**.

15834 `x` Matches an optionally signed hexadecimal integer, whose format is the same as
15835 expected for the subject sequence of *wcstoul*( ) with the value 16 for the *base* argument.
15836 In the absence of a size modifier, the application shall ensure that the corresponding
15837 argument is a pointer to **unsigned**.

15838 `a`, `e`, `f`, `g`
15839 Matches an optionally signed floating-point number, infinity, or NaN whose format is
15840 the same as expected for the subject sequence of *wcstod*( ). In the absence of a size
15841 modifier, the application shall ensure that the corresponding argument is a pointer to
15842 **float**.

15843 If the *fwprintf*( ) family of functions generates character string representations for
15844 infinity and NaN (a symbolic entity encoded in floating-point format) to support
15845 IEEE Std 754-1985, the *fwscanf*( ) family of functions shall recognize them as input.

15846 `s` Matches a sequence of non white-space wide characters. If no `l` (ell) qualifier is present,
15847 characters from the input field shall be converted as if by repeated calls to the
15848 *wcrtomb*( ) function, with the conversion state described by an **mbstate_t** object

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15849 initialized to zero before the first wide character is converted. The application shall
15850 ensure that the corresponding argument is a pointer to a character array large enough
15851 to accept the sequence and the terminating null character, which shall be added
15852 automatically.

15853 Otherwise, the application shall ensure that the corresponding argument is a pointer to
15854 an array of **wchar_t** large enough to accept the sequence and the terminating null wide
15855 character, which shall be added automatically.

15856 `[` Matches a non-empty sequence of wide characters from a set of expected wide
15857 characters (the *scanset*). If no `l` (ell) qualifier is present, wide characters from the input
15858 field shall be converted as if by repeated calls to the *wcrtomb*( ) function, with the
15859 conversion state described by an **mbstate_t** object initialized to zero before the first
15860 wide character is converted. The application shall ensure that the corresponding
15861 argument is a pointer to a character array large enough to accept the sequence and the
15862 terminating null character, which shall be added automatically.

15863 If an `l` (ell) qualifier is present, the application shall ensure that the corresponding
15864 argument is a pointer to an array of **wchar_t** large enough to accept the sequence and
15865 the terminating null wide character, which shall be added automatically.

15866 The conversion specification includes all subsequent wide characters in the *format*
15867 string up to and including the matching right square bracket (`']'`). The wide
15868 characters between the square brackets (the *scanlist*) comprise the scanset, unless the
15869 wide character after the left square bracket is a circumflex (`'^'`), in which case the
15870 scanset contains all wide characters that do not appear in the scanlist between the
15871 circumflex and the right square bracket. If the conversion specification begins with
15872 `"[]"` or `"[^]"`, the right square bracket is included in the scanlist and the next right
15873 square bracket is the matching right square bracket that ends the conversion
15874 specification; otherwise, the first right square bracket is the one that ends the
15875 conversion specification. If a `'-'` is in the scanlist and is not the first wide character,
15876 nor the second where the first wide character is a `'^'`, nor the last wide character, the
15877 behavior is implementation-defined.

15878 `c` Matches a sequence of wide characters of exactly the number specified by the field
15879 width (1 if no field width is present in the conversion specification).

15880 If no `l` (ell) length modifier is present, characters from the input field shall be converted
15881 as if by repeated calls to the *wcrtomb*( ) function, with the conversion state described by
15882 an **mbstate_t** object initialized to zero before the first wide character is converted. The
15883 corresponding argument shall be a pointer to the initial element of a character array
15884 large enough to accept the sequence. No null character is added.

15885 If an `l` (ell) length modifier is present, the corresponding argument shall be a pointer to
15886 the initial element of an array of **wchar_t** large enough to accept the sequence. No null
15887 wide character is added.

15888 Otherwise, the application shall ensure that the corresponding argument is a pointer to
15889 an array of **wchar_t** large enough to accept the sequence. No null wide character is
15890 added.

15891 `p` Matches an implementation-defined set of sequences, which shall be the same as the set
15892 of sequences that is produced by the `%p` conversion specification of the corresponding
15893 *fwprintf*( ) functions. The application shall ensure that the corresponding argument is a
15894 pointer to a pointer to **void**. The interpretation of the input item is implementation-
15895 defined. If the input item is a value converted earlier during the same program
15896 execution, the pointer that results shall compare equal to that value; otherwise, the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

behavior of the %p conversion is undefined.

n — No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which is to be written the number of wide characters read from the input so far by this call to the *fwscanf*( ) functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing wide character or a field width, the behavior is undefined.

XSI C — Equivalent to lc.

XSI S — Equivalent to ls.

% — Matches a single '%' wide character; no conversion or assignment shall occur. The complete conversion specification shall be %%.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively, a, e, f, g, and x.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current conversion specification (except for %n) have been read (other than leading white-space, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in *swscanf*( ) shall be equivalent to encountering end-of-file for *fwscanf*( ).

If conversion terminates on a conflicting input, the offending input shall be left unread in the input. Any trailing white space (including <newline>) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.

CX The *fwscanf*( ) and *wscanf*( ) functions may mark the *st_atime* field of the file associated with *stream* for update. The *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgetwc*( ), *fgets*( ), *fgetws*( ), *fread*( ), *getc*( ), *getwc*( ), *getchar*( ), *getwchar*( ), *gets*( ), *fscanf*( ), or *fwscanf*( ) using *stream* that returns data not supplied by a prior call to *ungetc*( ).

RETURN VALUE

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF shall be returned. If a read CX error occurs, the error indicator for the stream is set, EOF shall be returned, and *errno* shall be set to indicate the error.

ERRORS

For the conditions under which the *fwscanf*( ) functions shall fail and may fail, refer to *fgetwc*( ).

In addition, *fwscanf*( ) may fail if:

XSI [EILSEQ] Input byte sequence does not form a valid character.

XSI [EINVAL] There are insufficient arguments.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15939 **EXAMPLES**

15940 The call:

15941 `int i, n; float x; char name[50];`
15942 `n = wscanf(L"%d%f%s", &i, &x, name);`

15943 with the input line:

15944 `25 54.32E-1 Hamster`

15945 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string
15946 `"Hamster"`.

15947 The call:

15948 `int i; float x; char name[50];`
15949 `(void) wscanf(L"%2d%f%*d %[0123456789]", &i, &x, name);`

15950 with input:

15951 `56789 0123 56a72`

15952 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string `"56\0"` in *name*. The next call to
15953 *getchar*( ) shall return the character `'a'`.

15954 **APPLICATION USAGE**

15955 In format strings containing the `'%'` form of conversion specifications, each argument in the
15956 argument list is used exactly once.

15957 **RATIONALE**

15958 None.

15959 **FUTURE DIRECTIONS**

15960 None.

15961 **SEE ALSO**

15962 *getwc*( ), *fwprintf*( ), *setlocale*( ), *wcstod*( ), *wcstol*( ), *wcstoul*( ), *wcrtomb*( ), the Base Definitions
15963 volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

15964 **CHANGE HISTORY**

15965 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
15966 (E).

15967 **Issue 6**

15968 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

15969 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

15970 - The prototypes for *fwscanf*( ) and *swscanf*( ) are updated.
15971 - The DESCRIPTION is updated.
15972 - The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
15973 - The `a`, `A`, and `F` conversion characters are added.

15974 The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion
15975 specification’’ consistently.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15976 **NAME**

15977 gai_strerror — address and name information error description

15978 **SYNOPSIS**

15979 `#include <netdb.h>`

15980 `const char *gai_strerror(int ecode);`

15981 **DESCRIPTION**

15982 The *gai_strerror*( ) function shall return a text string describing an error value for the *getaddrinfo*( )
15983 and *getnameinfo*( ) functions listed in the **<netdb.h>** header.

15984 When the *ecode* argument is one of the following values listed in the **<netdb.h>** header:

15985 [EAI_AGAIN]
15986 [EAI_BADFLAGS]
15987 [EAI_FAIL]
15988 [EAI_FAMILY]
15989 [EAI_MEMORY]
15990 [EAI_NONAME]
15991 [EAI_OVERFLOW]
15992 [EAI_SERVICE]
15993 [EAI_SOCKTYPE]
15994 [EAI_SYSTEM]

15995 the function return value shall point to a string describing the error. If the argument is not one
15996 of those values, the function shall return a pointer to a string whose contents indicate an
15997 unknown error.

15998 **RETURN VALUE**

15999 Upon successful completion, *gai_strerror*( ) shall return a pointer to an implementation-defined
16000 string.

16001 **ERRORS**

16002 No errors are defined.

16003 **EXAMPLES**

16004 None.

16005 **APPLICATION USAGE**

16006 None.

16007 **RATIONALE**

16008 None.

16009 **FUTURE DIRECTIONS**

16010 None.

16011 **SEE ALSO**

16012 *getaddrinfo*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<netdb.h>**

16013 **CHANGE HISTORY**

16014 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

16015 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from
16016 **char *** to **const char ***. This is for coordination with the IPnG Working Group.

16017 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the
16018 [EAI_OVERFLOW] error code.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16019 **NAME**

16020 gcvt — convert a floating-point number to a string (**LEGACY**)

16021 **SYNOPSIS**

16022 XSI
```
#include <stdlib.h>
```

16023
```
char *gcvt(double value, int ndigit, char *buf);
```

16024

16025 **DESCRIPTION**

16026 Refer to *ecvt*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16027 **NAME**

16028 getaddrinfo — get address information

16029 **SYNOPSIS**

16030
```
#include <sys/socket.h>
```
16031
```
#include <netdb.h>
```

16032
```
int getaddrinfo(const char *restrict nodename,
    const char *restrict servname,
    const struct addrinfo *restrict hints,
    struct addrinfo **restrict res);
```

16036 **DESCRIPTION**

16037 Refer to *freeaddrinfo*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16038 **NAME**

16039 getc — get a byte from a stream

16040 **SYNOPSIS**

16041 `#include <stdio.h>`

16042 `int getc(FILE *stream);`

16043 **DESCRIPTION**

16044 CX The functionality described on this reference page is aligned with the ISO C standard. Any
16045 conflict between the requirements described here and the ISO C standard is unintentional. This
16046 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16047 The *getc*( ) function shall be equivalent to *fgetc*( ), except that if it is implemented as a macro it
16048 may evaluate *stream* more than once, so the argument should never be an expression with side
16049 effects.

16050 **RETURN VALUE**

16051 Refer to *fgetc*( ).

16052 **ERRORS**

16053 Refer to *fgetc*( ).

16054 **EXAMPLES**

16055 None.

16056 **APPLICATION USAGE**

16057 If the integer value returned by *getc*( ) is stored into a variable of type **char** and then compared
16058 against the integer constant EOF, the comparison may never succeed, because sign-extension of
16059 a variable of type **char** on widening to integer is implementation-defined.

16060 Since it may be implemented as a macro, *getc*( ) may treat incorrectly a *stream* argument with
16061 side effects. In particular, *getc*(*f*++) does not necessarily work as expected. Therefore, use of this
16062 function should be preceded by `"#undef getc"` in such situations; *fgetc*( ) could also be used.

16063 **RATIONALE**

16064 None.

16065 **FUTURE DIRECTIONS**

16066 None.

16067 **SEE ALSO**

16068 *fgetc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

16069 **CHANGE HISTORY**

16070 First released in Issue 1. Derived from Issue 1 of the SVID.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked — stdio with explicit client locking

SYNOPSIS

TSF

```
#include <stdio.h>

int getc_unlocked(FILE *stream);
int getchar_unlocked(void);
int putc_unlocked(int c, FILE *stream);
int putchar_unlocked(int c);
```

DESCRIPTION

Versions of the functions *getc*(), *getchar*(), *putc*(), and *putchar*() respectively named *getc_unlocked*(), *getchar_unlocked*(), *putc_unlocked*(), and *putchar_unlocked*() shall be provided which are functionally equivalent to the original versions, with the exception that they are not required to be implemented in a thread-safe manner. They may only safely be used within a scope protected by *flockfile*() (or *ftrylockfile*()) and *funlockfile*(). These functions may safely be used in a multi-threaded program if and only if they are called while the invoking thread owns the (**FILE ***) object, as is the case after a successful call to the *flockfile*() or *ftrylockfile*() functions.

RETURN VALUE

See *getc*(), *getchar*(), *putc*(), and *putchar*().

ERRORS

See *getc*(), *getchar*(), *putc*(), and *putchar*().

EXAMPLES

None.

APPLICATION USAGE

Since they may be implemented as macros, *getc_unlocked*() and *putc_unlocked*() may treat incorrectly a *stream* argument with side effects. In particular, *getc_unlocked*(*f++) and *putc_unlocked*(*f++) do not necessarily work as expected. Therefore, use of these functions in such situations should be preceded by the following statement as appropriate:

```
#undef getc_unlocked
#undef putc_unlocked
```

RATIONALE

Some I/O functions are typically implemented as macros for performance reasons (for example, *putc*() and *getc*()). For safety, they need to be synchronized, but it is often too expensive to synchronize on every character. Nevertheless, it was felt that the safety concerns were more important; consequently, the *getc*(), *getchar*(), *putc*(), and *putchar*() functions are required to be thread-safe. However, unlocked versions are also provided with names that clearly indicate the unsafe nature of their operation but can be used to exploit their higher performance. These unlocked versions can be safely used only within explicitly locked program regions, using exported locking primitives. In particular, a sequence such as:

```
flockfile(fileptr);
putc_unlocked('1', fileptr);
putc_unlocked('\n', fileptr);
fprintf(fileptr, "Line 2\n");
funlockfile(fileptr);
```

is permissible, and results in the text sequence:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
1
Line 2
```

being printed without being interspersed with output from other threads.

It would be wrong to have the standard names such as *getc*(), *putc*(), and so on, map to the ''faster, but unsafe'' rather than the ''slower, but safe'' versions. In either case, you would still want to inspect all uses of *getc*(), *putc*(), and so on, by hand when converting existing code. Choosing the safe bindings as the default, at least, results in correct code and maintains the ''atomicity at the function'' invariant. To do otherwise would introduce gratuitous synchronization errors into converted code. Other routines that modify the *stdio* (**FILE ***) structures or buffers are also safely synchronized.

Note that there is no need for functions of the form *getc_locked*(), *putc_locked*(), and so on, since this is the functionality of *getc*(), *putc*(), *et al*. It would be inappropriate to use a feature test macro to switch a macro definition of *getc*() between *getc_locked*() and *getc_unlocked*(), since the ISO C standard requires an actual function to exist, a function whose behavior could not be changed by the feature test macro. Also, providing both the *xxx_locked*() and *xxx_unlocked*() forms leads to the confusion of whether the suffix describes the behavior of the function or the circumstances under which it should be used.

Three additional routines, *flockfile*(), *ftrylockfile*(), and *funlockfile*() (which may be macros), are provided to allow the user to delineate a sequence of I/O statements that are executed synchronously.

The *ungetc*() function is infrequently called relative to the other functions/macros so no unlocked variation is needed.

FUTURE DIRECTIONS

None.

SEE ALSO

getc(), *getchar*(), *putc*(), *putchar*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

These functions are marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing how applications should be written to avoid the case when the functions are implemented as macros.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16151 **NAME**

16152 getchar — get a byte from a stdin stream

16153 **SYNOPSIS**

16154
```
#include <stdio.h>
```

16155
```
int getchar(void);
```

16156 **DESCRIPTION**

16157 CX The functionality described on this reference page is aligned with the ISO C standard. Any
16158 conflict between the requirements described here and the ISO C standard is unintentional. This
16159 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16160 The *getchar*( ) function shall be equivalent to *getc*(*stdin*).

16161 **RETURN VALUE**

16162 Refer to *fgetc*( ).

16163 **ERRORS**

16164 Refer to *fgetc*( ).

16165 **EXAMPLES**

16166 None.

16167 **APPLICATION USAGE**

16168 If the integer value returned by *getchar*( ) is stored into a variable of type **char** and then
16169 compared against the integer constant EOF, the comparison may never succeed, because sign-
16170 extension of a variable of type **char** on widening to integer is implementation-defined.

16171 **RATIONALE**

16172 None.

16173 **FUTURE DIRECTIONS**

16174 None.

16175 **SEE ALSO**

16176 *getc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

16177 **CHANGE HISTORY**

16178 First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16179 **NAME**

16180 getchar_unlocked — stdio with explicit client locking

16181 **SYNOPSIS**

16182 TSF
```
#include <stdio.h>
```

16183
```
int getchar_unlocked(void);
```

16184

16185 **DESCRIPTION**

16186 Refer to *getc_unlocked*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

-:`,,,`:-`:-`:,,:,,:,`:,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

16187 **NAME**

16188 getcontext, setcontext — get and set current user context

16189 **SYNOPSIS**

16190 XSI
```
#include <ucontext.h>

int getcontext(ucontext_t *ucp);
int setcontext(const ucontext_t *ucp);
```
16193

16194 **DESCRIPTION**

16195 The *getcontext*( ) function shall initialize the structure pointed to by *ucp* to the current user
16196 context of the calling thread. The **ucontext_t** type that *ucp* points to defines the user context and
16197 includes the contents of the calling thread's machine registers, the signal mask, and the current
16198 execution stack.

16199 The *setcontext*( ) function shall restore the user context pointed to by *ucp*. A successful call to
16200 *setcontext*( ) shall not return; program execution resumes at the point specified by the *ucp*
16201 argument passed to *setcontext*( ). The *ucp* argument should be created either by a prior call to
16202 *getcontext*( ) or *makecontext*( ), or by being passed as an argument to a signal handler. If the *ucp*
16203 argument was created with *getcontext*( ), program execution continues as if the corresponding
16204 call of *getcontext*( ) had just returned. If the *ucp* argument was created with *makecontext*( ),
16205 program execution continues with the function passed to *makecontext*( ). When that function
16206 returns, the thread shall continue as if after a call to *setcontext*( ) with the *ucp* argument that was
16207 input to *makecontext*( ). If the *uc_link* member of the **ucontext_t** structure pointed to by the *ucp*
16208 argument is equal to 0, then this context is the main context, and the thread shall exit when this
16209 context returns. The effects of passing a *ucp* argument obtained from any other source are
16210 unspecified.

16211 **RETURN VALUE**

16212 Upon successful completion, *setcontext*( ) shall not return and *getcontext*( ) shall return 0;
16213 otherwise, a value of –1 shall be returned.

16214 **ERRORS**

16215 No errors are defined.

16216 **EXAMPLES**

16217 Refer to *makecontext*( ).

16218 **APPLICATION USAGE**

16219 When a signal handler is executed, the current user context is saved and a new context is
16220 created. If the thread leaves the signal handler via *longjmp*( ), then it is unspecified whether the
16221 context at the time of the corresponding *setjmp*( ) call is restored and thus whether future calls to
16222 *getcontext*( ) provide an accurate representation of the current context, since the context restored
16223 by *longjmp*( ) does not necessarily contain all the information that *setcontext*( ) requires. Signal
16224 handlers should use *siglongjmp*( ) or *setcontext*( ) instead.

16225 Conforming applications should not modify or access the *uc_mcontext* member of **ucontext_t**. A
16226 conforming application cannot assume that context includes any process-wide static data,
16227 possibly including *errno*. Users manipulating contexts should take care to handle these
16228 explicitly when required.

16229 Use of contexts to create alternate stacks is not defined by this volume of IEEE Std 1003.1-2001.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16230 **RATIONALE**

16231 None.

16232 **FUTURE DIRECTIONS**

16233 None.

16234 **SEE ALSO**

16235 *bsd_signal*( ), *makecontext*( ), *setcontext*( ), *setjmp*( ), *sigaction*( ), *sigaltstack*( ), *siglongjmp*( ),
16236 *sigprocmask*( ), *sigsetjmp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<ucontext.h>**

16237 **CHANGE HISTORY**

16238 First released in Issue 4, Version 2.

16239 **Issue 5**

16240 Moved from X/OPEN UNIX extension to BASE.

16241 The following sentence was removed from the DESCRIPTION: ‘‘If the *ucp* argument was passed
16242 to a signal handler, program execution continues with the program instruction following the
16243 instruction interrupted by the signal.’’

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

16244 **NAME**

16245 getcwd — get the pathname of the current working directory

16246 **SYNOPSIS**

16247 ```
#include <unistd.h>
```

16248 ```
char *getcwd(char *buf, size_t size);
```

16249 **DESCRIPTION**

16250 The *getcwd*( ) function shall place an absolute pathname of the current working directory in the
16251 array pointed to by *buf*, and return *buf*. The pathname copied to the array shall contain no
16252 components that are symbolic links. The *size* argument is the size in bytes of the character array
16253 pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd*( ) is unspecified.

16254 **RETURN VALUE**

16255 Upon successful completion, *getcwd*( ) shall return the *buf* argument. Otherwise, *getcwd*( ) shall
16256 return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by
16257 *buf* are then undefined.

16258 **ERRORS**

16259 The *getcwd*( ) function shall fail if:

16260 [EINVAL] The *size* argument is 0.

16261 [ERANGE] The *size* argument is greater than 0, but is smaller than the length of the
16262 pathname +1.

16263 The *getcwd*( ) function may fail if:

16264 [EACCES] Read or search permission was denied for a component of the pathname.

16265 [ENOMEM] Insufficient storage space is available.

16266 **EXAMPLES**

16267 **Determining the Absolute Pathname of the Current Working Directory**

16268 The following example returns a pointer to an array that holds the absolute pathname of the
16269 current working directory. The pointer is returned in the *ptr* variable, which points to the *buf*
16270 array where the pathname is stored.

16271 ```
#include <stdlib.h>
16272 #include <unistd.h>
16273 ...
16274 long size;
16275 char *buf;
16276 char *ptr;
```

16277 ```
size = pathconf(".", _PC_PATH_MAX);
```

16278 ```
if ((buf = (char *)malloc((size_t)size)) != NULL)
16279   ptr = getcwd(buf, (size_t)size);
16280 ...
```

16281 **APPLICATION USAGE**

16282 None.



Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale
```

16283 **RATIONALE**

16284 Since the maximum pathname length is arbitrary unless {PATH_MAX} is defined, an application
16285 generally cannot supply a *buf* with *size* {{PATH_MAX}+1}.

16286 Having *getcwd*( ) take no arguments and instead use the *malloc*( ) function to produce space for
16287 the returned argument was considered. The advantage is that *getcwd*( ) knows how big the
16288 working directory pathname is and can allocate an appropriate amount of space. But the
16289 programmer would have to use the *free*( ) function to free the resulting object, or each use of
16290 *getcwd*( ) would further reduce the available memory. Also, *malloc*( ) and *free*( ) are used nowhere
16291 else in this volume of IEEE Std 1003.1-2001. Finally, *getcwd*( ) is taken from the SVID where it has
16292 the two arguments used in this volume of IEEE Std 1003.1-2001.

16293 The older function *getwd*( ) was rejected for use in this context because it had only a buffer
16294 argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except
16295 to depend on the programmer to provide a large enough buffer.

16296 On some implementations, if *buf* is a null pointer, *getcwd*( ) may obtain *size* bytes of memory
16297 using *malloc*( ). In this case, the pointer returned by *getcwd*( ) may be used as the argument in a
16298 subsequent call to *free*( ). Invoking *getcwd*( ) with *buf* as a null pointer is not recommended in
16299 conforming applications.

16300 If a program is operating in a directory where some (grand)parent directory does not permit
16301 reading, *getcwd*( ) may fail, as in most implementations it must read the directory to determine
16302 the name of the file. This can occur if search, but not read, permission is granted in an
16303 intermediate directory, or if the program is placed in that directory by some more privileged
16304 process (for example, login). Including the [EACCES] error condition makes the reporting of the
16305 error consistent and warns the application writer that *getcwd*( ) can fail for reasons beyond the
16306 control of the application writer or user. Some implementations can avoid this occurrence (for
16307 example, by implementing *getcwd*( ) using *pwd*, where *pwd* is a set-user-root process), thus the
16308 error was made optional. Since this volume of IEEE Std 1003.1-2001 permits the addition of other
16309 errors, this would be a common addition and yet one that applications could not be expected to
16310 deal with without this addition.

16311 **FUTURE DIRECTIONS**

16312 None.

16313 **SEE ALSO**

16314 *malloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

16315 **CHANGE HISTORY**

16316 First released in Issue 1. Derived from Issue 1 of the SVID.

16317 **Issue 6**

16318 The following new requirements on POSIX implementations derive from alignment with the
16319 Single UNIX Specification:

16320 - The [ENOMEM] optional error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16321 **NAME**

16322 getdate — convert user format date and time

16323 **SYNOPSIS**

16324 XSI
```
#include <time.h>

struct tm *getdate(const char *string);
```
16325

16326

16327 **DESCRIPTION**

16328 The *getdate*( ) function shall convert a string representation of a date or time into a broken-down
16329 time.

16330 The external variable or macro *getdate_err* is used by *getdate*( ) to return error values.

16331 Templates are used to parse and interpret the input string. The templates are contained in a text
16332 file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to
16333 indicate the full pathname of the file that contains the templates. The first line in the template
16334 that matches the input specification is used for interpretation and conversion into the internal
16335 time format.

16336 The following conversion specifications shall be supported:

16337 `%%` Equivalent to `%`.

16338 `%a` Abbreviated weekday name.

16339 `%A` Full weekday name.

16340 `%b` Abbreviated month name.

16341 `%B` Full month name.

16342 `%c` Locale's appropriate date and time representation.

16343 `%C` Century number [00,99]; leading zeros are permitted but not required.

16344 `%d` Day of month [01,31]; the leading 0 is optional.

16345 `%D` Date as `%m/%d/%y`.

16346 `%e` Equivalent to `%d`.

16347 `%h` Abbreviated month name.

16348 `%H` Hour [00,23].

16349 `%I` Hour [01,12].

16350 `%m` Month number [01,12].

16351 `%M` Minute [00,59].

16352 `%n` Equivalent to <newline>.

16353 `%p` Locale's equivalent of either AM or PM.

16354 `%r` The locale's appropriate representation of time in AM and PM notation. In the POSIX
16355 locale, this shall be equivalent to `%I:%M:%S %p`.

16356 `%R` Time as `%H:%M`.

16357 `%S` Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap
16358 seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap
16359 second data must come from some external source.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16360 `%t` Equivalent to <tab>.

16361 `%T` Time as `%H:%M:%S`.

16362 `%w` Weekday number (Sunday = [0,6]).

16363 `%x` Locale's appropriate date representation.

16364 `%X` Locale's appropriate time representation.

16365 `%y` Year within century. When a century is not otherwise specified, values in the range
16366 [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall
16367 refer to years 2000 to 2068 inclusive.

16368 **Note:** It is expected that in a future version of IEEE Std 1003.1-2001 the default century
16369 inferred from a 2-digit year will change. (This would apply to all commands
16370 accepting a 2-digit year as input.)

16371 `%Y` Year as `"ccyy"` (for example, 2001).

16372 `%Z` Timezone name or no characters if no timezone exists. If the timezone supplied by `%Z` is
16373 not the timezone that *getdate*( ) expects, an invalid input specification error shall result.
16374 The *getdate*( ) function calculates an expected timezone based on information supplied
16375 to the function (such as the hour, day, and month).

16376 The match between the template and input specification performed by *getdate*( ) shall be case-
16377 insensitive.

16378 The month and weekday names can consist of any combination of upper and lowercase letters.
16379 The process can request that the input date or time specification be in a specific language by
16380 setting the *LC_TIME* category (see *setlocale*( )).

16381 Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most
16382 two digits are allowed for those descriptors, including leading zeros. Extra whitespace in either
16383 the template file or in *string* shall be ignored.

16384 The results are undefined if the conversion specifications `%c`, `%x`, and `%X` include unsupported
16385 conversion specifications.

16386 The following rules apply for converting the input specification into the internal format:

16387 • If `%Z` is being scanned, then *getdate*( ) shall initialize the broken-down time to be the current
16388 time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on
16389 the current local time as if *localtime*( ) had been called.

16390 • If only the weekday is given, the day chosen shall be the day, starting with today and moving
16391 into the future, which first matches the named day.

16392 • If only the month (and no year) is given, the month chosen shall be the month, starting with
16393 the current month and moving into the future, which first matches the named month. The
16394 first day of the month shall be assumed if no day is given.

16395 • If no hour, minute, and second are given, the current hour, minute, and second shall be
16396 assumed.

16397 • If no date is given, the hour chosen shall be the hour, starting with the current hour and
16398 moving into the future, which first matches the named hour.

16399 If a conversion specification in the DATEMSK file does not correspond to one of the conversion
16400 specifications above, the behavior is unspecified.

16401 The *getdate*( ) function need not be reentrant. A function that is not required to be reentrant is not
16402 required to be thread-safe.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16403 **RETURN VALUE**

16404 Upon successful completion, *getdate*( ) shall return a pointer to a **struct tm**. Otherwise, it shall
16405 return a null pointer and set *getdate_err* to indicate the error.

16406 **ERRORS**

16407 The *getdate*( ) function shall fail in the following cases, setting *getdate_err* to the value shown in
16408 the list below. Any changes to *errno* are unspecified.

16409 1. The *DATEMSK* environment variable is null or undefined.

16410 2. The template file cannot be opened for reading.

16411 3. Failed to get file status information.

16412 4. The template file is not a regular file.

16413 5. An I/O error is encountered while reading the template file.

16414 6. Memory allocation failed (not enough memory available).

16415 7. There is no line in the template that matches the input.

16416 8. Invalid input specification. For example, February 31; or a time is specified that cannot be
16417 represented in a **time_t** (representing the time in seconds since the Epoch).

16418 **EXAMPLES**

16419 1. The following example shows the possible contents of a template:

```
16420 %m
16421 %A %B %d, %Y, %H:%M:%S
16422 %A
16423 %B
16424 %m/%d/%y %I %p
16425 %d,%m,%Y %H:%M
16426 at %A the %dst of %B in %Y
16427 run job at %I %p,%B %dnd
16428 %A den %d. %B %Y %H.%M Uhr
```

16429 2. The following are examples of valid input specifications for the template in Example 1:

```
16430 getdate("10/1/87 4 PM");
16431 getdate("Friday");
16432 getdate("Friday September 18, 1987, 10:30:30");
16433 getdate("24,9,1986 10:30");
16434 getdate("at monday the 1st of december in 1986");
16435 getdate("run job at 3 PM, december 2nd");
```

16436 If the *LC_TIME* category is set to a German locale that includes *freitag* as a weekday name
16437 and *oktober* as a month name, the following would be valid:

```
16438 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

16439 3. The following example shows how local date and time specification can be defined in the
16440 template:

-:`,,`-:`:`,,`,,`,`,,`:---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Invocation	Line in Template
`getdate("11/27/86")`	`%m/%d/%y`
`getdate("27.11.86")`	`%d.%m.%y`
`getdate("86-11-27")`	`%y-%m-%d`
`getdate("Friday 12:00:00")`	`%A %H:%M:%S`

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC_TIME* category is set to the default C locale:

Input	Line in Template	Date
`Mon`	`%a`	Mon Sep 22 12:19:47 EDT 1986
`Sun`	`%a`	Sun Sep 28 12:19:47 EDT 1986
`Fri`	`%a`	Fri Sep 26 12:19:47 EDT 1986
`September`	`%B`	Mon Sep 1 12:19:47 EDT 1986
`January`	`%B`	Thu Jan 1 12:19:47 EST 1987
`December`	`%B`	Mon Dec 1 12:19:47 EST 1986
`Sep Mon`	`%b %a`	Mon Sep 1 12:19:47 EDT 1986
`Jan Fri`	`%b %a`	Fri Jan 2 12:19:47 EST 1987
`Dec Mon`	`%b %a`	Mon Dec 1 12:19:47 EST 1986
`Jan Wed 1989`	`%b %a %Y`	Wed Jan 4 12:19:47 EST 1989
`Fri 9`	`%a %H`	Fri Sep 26 09:00:00 EDT 1986
`Feb 10:30`	`%b %H:%S`	Sun Feb 1 10:00:30 EST 1987
`10:30`	`%H:%M`	Tue Sep 23 10:30:00 EDT 1986
`13:30`	`%H:%M`	Mon Sep 22 13:30:00 EDT 1986

APPLICATION USAGE

Although historical versions of *getdate*() did not require that **<time.h>** declare the external variable *getdate_err*, this volume of IEEE Std 1003.1-2001 does require it. The standard developers encourage applications to remove declarations of *getdate_err* and instead incorporate the declaration by including **<time.h>**.

Applications should use `%Y` (4-digit years) in preference to `%y` (2-digit years).

RATIONALE

In standard locales, the conversion specifications `%c`, `%x`, and `%X` do not include unsupported conversion specifiers and so the text regarding results being undefined is not a problem in that case.

FUTURE DIRECTIONS

None.

SEE ALSO

ctime(), *localtime*(), *setlocale*(), *strftime*(), *times*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The last paragraph of the DESCRIPTION is added.

The `%C` conversion specification is added, and the exact meaning of the `%y` conversion specification is clarified in the DESCRIPTION.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

The `%R` conversion specification is changed to follow historical practice.

Issue 6

The DESCRIPTION is updated to refer to ''seconds since the Epoch'' rather than ''seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970'' for consistency with other *time* functions.

The description of `%S` is updated so that the valid range is [00,60] rather than [00,61].

The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors for consistency with other functions.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16496 **NAME**

16497 getegid — get the effective group ID

16498 **SYNOPSIS**

16499
```
#include <unistd.h>
```

16500
```
gid_t getegid(void);
```

16501 **DESCRIPTION**

16502 The *getegid*( ) function shall return the effective group ID of the calling process.

16503 **RETURN VALUE**

16504 The *getegid*( ) function shall always be successful and no return value is reserved to indicate an
16505 error.

16506 **ERRORS**

16507 No errors are defined.

16508 **EXAMPLES**

16509 None.

16510 **APPLICATION USAGE**

16511 None.

16512 **RATIONALE**

16513 None.

16514 **FUTURE DIRECTIONS**

16515 None.

16516 **SEE ALSO**

16517 *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( ), the Base
16518 Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

16519 **CHANGE HISTORY**

16520 First released in Issue 1. Derived from Issue 1 of the SVID.

16521 **Issue 6**

16522 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

16523 The following new requirements on POSIX implementations derive from alignment with the
16524 Single UNIX Specification:

16525 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
16526 required for conforming implementations of previous POSIX specifications, it was not
16527 required for UNIX applications.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16528 **NAME**

16529 getenv — get value of an environment variable

16530 **SYNOPSIS**

16531
```
#include <stdlib.h>
```

16532
```
char *getenv(const char *name);
```

16533 **DESCRIPTION**

16534 CX The functionality described on this reference page is aligned with the ISO C standard. Any
16535 conflict between the requirements described here and the ISO C standard is unintentional. This
16536 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16537 The *getenv*( ) function shall search the environment of the calling process (see the Base
16538 Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables) for the
16539 environment variable *name* if it exists and return a pointer to the value of the environment
16540 variable. If the specified environment variable cannot be found, a null pointer shall be returned.
16541 The application shall ensure that it does not modify the string pointed to by the *getenv*( )
16542 function.

16543 CX The string pointed to may be overwritten by a subsequent call to *getenv*( ), *setenv*( ), or *unsetenv*( ),
16544 but shall not be overwritten by a call to any other function in this volume of
16545 IEEE Std 1003.1-2001.

16546 CX If the application modifies *environ* or the pointers to which it points, the behavior of *getenv*( ) is
16547 undefined.

16548 The *getenv*( ) function need not be reentrant. A function that is not required to be reentrant is not
16549 required to be thread-safe.

16550 **RETURN VALUE**

16551 Upon successful completion, *getenv*( ) shall return a pointer to a string containing the *value* for
16552 the specified *name*. If the specified *name* cannot be found in the environment of the calling
16553 process, a null pointer shall be returned.

16554 The return value from *getenv*( ) may point to static data which may be overwritten by
16555 CX subsequent calls to *getenv*( ), *setenv*( ), or *unsetenv*( ).

16556 XSI On XSI-conformant systems, the return value from *getenv*( ) may point to static data which may
16557 also be overwritten by subsequent calls to *putenv*( ).

16558 **ERRORS**

16559 No errors are defined.

16560 **EXAMPLES**

16561 **Getting the Value of an Environment Variable**

16562 The following example gets the value of the *HOME* environment variable.

16563
```
#include <stdlib.h>
```
16564
```
...
```
16565
```
const char *name = "HOME";
char *value;
```
16566

16567
```
value = getenv(name);
```


--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

APPLICATION USAGE

None.

RATIONALE

The *clearenv*( ) function was considered but rejected. The *putenv*( ) function has now been included for alignment with the Single UNIX Specification.

The *getenv*( ) function is inherently not reentrant because it returns a value pointing to static data.

Conforming applications are required not to modify *environ* directly, but to use only the functions described here to manipulate the process environment as an abstract object. Thus, the implementation of the environment access functions has complete control over the data structure used to represent the environment (subject to the requirement that *environ* be maintained as a list of strings with embedded equal signs for applications that wish to scan the environment). This constraint allows the implementation to properly manage the memory it allocates, either by using allocated storage for all variables (copying them on the first invocation of *setenv*( ) or *unsetenv*( )), or keeping track of which strings are currently in allocated space and which are not, via a separate table or some other means. This enables the implementation to free any allocated space used by strings (and perhaps the pointers to them) stored in *environ* when *unsetenv*( ) is called. A C runtime start-up procedure (that which invokes *main*( ) and perhaps initializes *environ*) can also initialize a flag indicating that none of the environment has yet been copied to allocated storage, or that the separate table has not yet been initialized.

In fact, for higher performance of *getenv*( ), the implementation could also maintain a separate copy of the environment in a data structure that could be searched much more quickly (such as an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when *setenv*( ) or *unsetenv*( ) is invoked.

Performance of *getenv*( ) can be important for applications which have large numbers of environment variables. Typically, applications like this use the environment as a resource database of user-configurable parameters. The fact that these variables are in the user's shell environment usually means that any other program that uses environment variables (such as *ls*, which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC_ALL*, and so on) is similarly slowed down by the linear search through the variables.

An implementation that maintains separate data structures, or even one that manages the memory it consumes, is not currently required as it was thought it would reduce consensus among implementors who do not want to change their historical implementations.

The POSIX Threads Extension states that multi-threaded applications must not modify *environ* directly, and that IEEE Std 1003.1-2001 is providing functions which such applications can use in the future to manipulate the environment in a thread-safe manner. Thus, moving away from application use of *environ* is desirable from that standpoint as well.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *putenv*( ), *setenv*( ), *unsetenv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16612 **Issue 5**

16613 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
16614 VALUE section.

16615 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

16616 **Issue 6**

16617 The following changes were made to align with the IEEE P1003.1a draft standard:

16618 • References added to the new *setenv*( ) and *unsetenv*( ) functions.

16619 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16620 **NAME**

16621 geteuid — get the effective user ID

16622 **SYNOPSIS**

16623
```
#include <unistd.h>
```

16624
```
uid_t geteuid(void);
```

16625 **DESCRIPTION**

16626 The *geteuid*( ) function shall return the effective user ID of the calling process.

16627 **RETURN VALUE**

16628 The *geteuid*( ) function shall always be successful and no return value is reserved to indicate an
16629 error.

16630 **ERRORS**

16631 No errors are defined.

16632 **EXAMPLES**

16633 None.

16634 **APPLICATION USAGE**

16635 None.

16636 **RATIONALE**

16637 None.

16638 **FUTURE DIRECTIONS**

16639 None.

16640 **SEE ALSO**

16641 *getegid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( ), the Base
16642 Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

16643 **CHANGE HISTORY**

16644 First released in Issue 1. Derived from Issue 1 of the SVID.

16645 **Issue 6**

16646 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

16647 The following new requirements on POSIX implementations derive from alignment with the
16648 Single UNIX Specification:

16649 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
16650 required for conforming implementations of previous POSIX specifications, it was not
16651 required for UNIX applications.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16652 **NAME**

16653 getgid — get the real group ID

16654 **SYNOPSIS**

16655
```
#include <unistd.h>
```

16656
```
gid_t getgid(void);
```

16657 **DESCRIPTION**

16658 The *getgid*( ) function shall return the real group ID of the calling process.

16659 **RETURN VALUE**

16660 The *getgid*( ) function shall always be successful and no return value is reserved to indicate an
16661 error.

16662 **ERRORS**

16663 No errors are defined.

16664 **EXAMPLES**

16665 None.

16666 **APPLICATION USAGE**

16667 None.

16668 **RATIONALE**

16669 None.

16670 **FUTURE DIRECTIONS**

16671 None.

16672 **SEE ALSO**

16673 *getegid*( ), *geteuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( ), the Base
16674 Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

16675 **CHANGE HISTORY**

16676 First released in Issue 1. Derived from Issue 1 of the SVID.

16677 **Issue 6**

16678 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

16679 The following new requirements on POSIX implementations derive from alignment with the
16680 Single UNIX Specification:

16681 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
16682 required for conforming implementations of previous POSIX specifications, it was not
16683 required for UNIX applications.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16684 **NAME**

16685 getgrent — get the group database entry

16686 **SYNOPSIS**

16687 XSI
```
#include <grp.h>
```

16688
```
struct group *getgrent(void);
```

16689

16690 **DESCRIPTION**

16691 Refer to *endgrent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

getgrgid, getgrgid_r — get group database entry for a group ID

SYNOPSIS

```
#include <grp.h>

struct group *getgrgid(gid_t gid);
int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
    size_t bufsize, struct group **result);
```

DESCRIPTION

The *getgrgid*( ) function shall search the group database for an entry with a matching *gid*.

The *getgrgid*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

TSF The *getgrgid_r*( ) function shall update the **group** structure pointed to by *grp* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the group database with a matching *gid*. Storage referenced by the group structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The maximum size needed for this buffer can be determined with the {_SC_GETGR_R_SIZE_MAX} *sysconf*( ) parameter. A NULL pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

Upon successful completion, *getgrgid*( ) shall return a pointer to a **struct group** with the structure defined in **<grp.h>** with a matching entry if one is found. The *getgrgid*( ) function shall return a null pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be set to indicate the error.

The return value may point to a static area which is overwritten by a subsequent call to *getgrent*( ), *getgrgid*( ), or *getgrnam*( ).

TSF If successful, the *getgrgid_r*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *getgrgid*( ) and *getgrgid_r*( ) functions may fail if:

[EIO]	An I/O error has occurred.
[EINTR]	A signal was caught during *getgrgid*( ).
[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
[ENFILE]	The maximum allowable number of files is currently open in the system.

TSF The *getgrgid_r*( ) function may fail if:

TSF [ERANGE]	Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **group** structure.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Finding an Entry in the Group Database

The following example uses *getgrgid*( ) to search the group database for a group ID that was previously stored in a **stat** structure, then prints out the group name if it is found. If the group is not found, the program prints the numeric value of the group for the entry.

```
#include <sys/types.h>
#include <grp.h>
#include <stdio.h>
...
struct stat statbuf;
struct group *grp;
...
if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8d", statbuf.st_gid);
...
```

APPLICATION USAGE

Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid*( ). If *errno* is set on return, an error occurred.

The *getgrgid_r*( ) function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent( ), *getgrnam*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<grp.h>**, **<limits.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getgrgid_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getgrgid*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getgrgid_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *gid*.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16783 **NAME**

16784 getgrnam, getgrnam_r — search group database for a name

16785 **SYNOPSIS**

16786 ```
#include <grp.h>
```

16787 ```
struct group *getgrnam(const char *name);
```

16788 TSF ```
int getgrnam_r(const char *name, struct group *grp, char *buffer,
16789   size_t bufsize, struct group **result);
```

16790

16791 **DESCRIPTION**

16792 The *getgrnam*( ) function shall search the group database for an entry with a matching *name*.

16793 The *getgrnam*( ) function need not be reentrant. A function that is not required to be reentrant is
16794 not required to be thread-safe.

16795 TSF The *getgrnam_r*( ) function shall update the **group** structure pointed to by *grp* and store a pointer
16796 to that structure at the location pointed to by *result*. The structure shall contain an entry from
16797 the group database with a matching *gid* or *name*. Storage referenced by the **group** structure is
16798 allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The
16799 maximum size needed for this buffer can be determined with the {_SC_GETGR_R_SIZE_MAX}
16800 *sysconf*( ) parameter. A NULL pointer is returned at the location pointed to by *result* on error or if
16801 the requested entry is not found.

16802 **RETURN VALUE**

16803 The *getgrnam*( ) function shall return a pointer to a **struct group** with the structure defined in
16804 **<grp.h>** with a matching entry if one is found. The *getgrnam*( ) function shall return a null
16805 pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be
16806 set to indicate the error.

16807 The return value may point to a static area which is overwritten by a subsequent call to
16808 *getgrent*( ), *getgrgid*( ), or *getgrnam*( ).

16809 TSF If successful, the *getgrnam_r*( ) function shall return zero; otherwise, an error number shall be
16810 returned to indicate the error.

16811 **ERRORS**

16812 The *getgrnam*( ) and *getgrnam_r*( ) functions may fail if:

16813 [EIO] An I/O error has occurred.

16814 [EINTR] A signal was caught during *getgrnam*( ).

16815 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

16816 [ENFILE] The maximum allowable number of files is currently open in the system.

16817 The *getgrnam_r*( ) function may fail if:

16818 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to
16819 be referenced by the resulting **group** structure.


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
```

EXAMPLES

None.

APPLICATION USAGE

Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam*( ). If *errno* is set on return, an error occurred.

The *getgrnam_r*( ) function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent( ), *getgrgid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<grp.h>**, **<limits.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getgrnam_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getgrnam*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getgrnam_r*( ) function is marked as part of the Thread-Safe Functions option.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16857 **NAME**

16858 getgroups — get supplementary group IDs

16859 **SYNOPSIS**

16860 `#include <unistd.h>`

16861 `int getgroups(int gidsetsize, gid_t grouplist[]);`

16862 **DESCRIPTION**

16863 The *getgroups*( ) function shall fill in the array *grouplist* with the current supplementary group
16864 IDs of the calling process. It is implementation-defined whether *getgroups*( ) also returns the
16865 effective group ID in the *grouplist* array.

16866 The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual
16867 number of group IDs stored in the array shall be returned. The values of array entries with
16868 indices greater than or equal to the value returned are undefined.

16869 If *gidsetsize* is 0, *getgroups*( ) shall return the number of group IDs that it would otherwise return
16870 without modifying the array pointed to by *grouplist*.

16871 If the effective group ID of the process is returned with the supplementary group IDs, the value
16872 returned shall always be greater than or equal to one and less than or equal to the value of
16873 {NGROUPS_MAX}+1.

16874 **RETURN VALUE**

16875 Upon successful completion, the number of supplementary group IDs shall be returned. A
16876 return value of –1 indicates failure and *errno* shall be set to indicate the error.

16877 **ERRORS**

16878 The *getgroups*( ) function shall fail if:

16879 [EINVAL] The *gidsetsize* argument is non-zero and less than the number of group IDs
16880 that would have been returned.

16881 **EXAMPLES**

16882 **Getting the Supplementary Group IDs of the Calling Process**

16883 The following example places the current supplementary group IDs of the calling process into
16884 the *group* array.

```
16885 #include <sys/types.h>
16886 #include <unistd.h>
16887 ...
16888 gid_t *group;
16889 int nogroups;
16890 long ngroups_max;

16891 ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
16892 group = (gid_t *)malloc(ngroups_max *sizeof(gid_t));

16893 ngroups = getgroups(ngroups_max, group);
```

16894 **APPLICATION USAGE**

16895 None.

16896 **RATIONALE**

16897 The related function *setgroups*( ) is a privileged operation and therefore is not covered by this
16898 volume of IEEE Std 1003.1-2001.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

As implied by the definition of supplementary groups, the effective group ID may appear in the array returned by *getgroups*( ) or it may be returned only by *getegid*( ). Duplication may exist, but the application needs to call *getegid*( ) to be sure of getting all of the information. Various implementation variations and administrative sequences cause the set of groups appearing in the result of *getgroups*( ) to vary in order and as to whether the effective group ID is included, even when the set of groups is the same (in the mathematical sense of ''set''). (The history of a process and its parents could affect the details of the result.)

Application writers should note that {NGROUPS_MAX} is not necessarily a constant on all implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *setgid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- A return value of 0 is not permitted, because {NGROUPS_MAX} cannot be 0. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added that the effective group ID may be included in the supplementary group list.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

16930 **NAME**

16931 gethostbyaddr, gethostbyname — network host database functions

16932 **SYNOPSIS**

16933 ```
#include <netdb.h>
```

16934 OB ```
struct hostent *gethostbyaddr(const void *addr, socklen_t len,
16935     int type);
16936 struct hostent *gethostbyname(const char *name);
```

16937

16938 **DESCRIPTION**

16939 These functions shall retrieve information about hosts. This information is considered to be
16940 stored in a database that can be accessed sequentially or randomly. Implementation of this
16941 database is unspecified.

16942 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,
16943 RFC 1035, and RFC 1886.

16944 Entries shall be returned in **hostent** structures.

16945 The *gethostbyaddr*( ) function shall return an entry containing addresses of address family *type* for
16946 the host with address *addr*. The *len* argument contains the length of the address pointed to by
16947 *addr*. The *gethostbyaddr*( ) function need not be reentrant. A function that is not required to be
16948 reentrant is not required to be thread-safe.

16949 The *gethostbyname*( ) function shall return an entry containing addresses of address family
16950 AF_INET for the host with name *name*. The *gethostbyname*( ) function need not be reentrant. A
16951 function that is not required to be reentrant is not required to be thread-safe.

16952 The *addr* argument of *gethostbyaddr*( ) shall be an **in_addr** structure when *type* is AF_INET. It
16953 contains a binary format (that is, not null-terminated) address in network byte order. The
16954 *gethostbyaddr*( ) function is not guaranteed to return addresses of address families other than
16955 AF_INET, even when such addresses exist in the database.

16956 If *gethostbyaddr*( ) returns successfully, then the *h_addrtype* field in the result shall be the same as
16957 the *type* argument that was passed to the function, and the *h_addr_list* field shall list a single
16958 address that is a copy of the *addr* argument that was passed to the function.

16959 The *name* argument of *gethostbyname*( ) shall be a node name; the behavior of *gethostbyname*( )
16960 when passed a numeric address string is unspecified. For IPv4, a numeric address string shall be
16961 in the dotted-decimal notation described in *inet_addr*( ).

16962 If *name* is not a numeric address string and is an alias for a valid host name, then *gethostbyname*( )
16963 shall return information about the host name to which the alias refers, and *name* shall be
16964 included in the list of aliases returned.

16965 **RETURN VALUE**

16966 Upon successful completion, these functions shall return a pointer to a **hostent** structure if the
16967 requested entry was found, and a null pointer if the end of the database was reached or the
16968 requested entry was not found.

16969 Upon unsuccessful completion, *gethostbyaddr*( ) and *gethostbyname*( ) shall set *h_errno* to indicate
16970 the error.

16971 **ERRORS**

16972 These functions shall fail in the following cases. The *gethostbyaddr*( ) and *gethostbyname*( )
16973 functions shall set *h_errno* to the value shown in the list below. Any changes to *errno* are
16974 unspecified.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16975 [HOST_NOT_FOUND]
16976 No such host is known.

16977 [NO_DATA] The server recognized the request and the name, but no address is available.
16978 Another type of request to the name server for the domain might return an
16979 answer.

16980 [NO_RECOVERY]
16981 An unexpected server failure occurred which cannot be recovered.

16982 [TRY_AGAIN] A temporary and possibly transient error occurred, such as a failure of a
16983 server to respond.

16984 **EXAMPLES**

16985 None.

16986 **APPLICATION USAGE**

16987 The *gethostbyaddr*( ) and *gethostbyname*( ) functions may return pointers to static data, which may
16988 be overwritten by subsequent calls to any of these functions.

16989 The *getaddrinfo*( ) and *getnameinfo*( ) functions are preferred over the *gethostbyaddr*( ) and
16990 *gethostbyname*( ) functions.

16991 **RATIONALE**

16992 None.

16993 **FUTURE DIRECTIONS**

16994 The *gethostbyaddr*( ) and *gethostbyname*( ) functions may be withdrawn in a future version.

16995 **SEE ALSO**

16996 *endhostent*( ), *endservent*( ), *gai_strerror*( ), *getaddrinfo*( ), *h_errno*, *inet_addr*( ), the Base Definitions
16997 volume of IEEE Std 1003.1-2001, **<netdb.h>**

16998 **CHANGE HISTORY**

16999 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17000 **NAME**

17001 gethostent — network host database functions

17002 **SYNOPSIS**

17003 `#include <netdb.h>`

17004 `struct hostent *gethostent(void);`

17005 **DESCRIPTION**

17006 Refer to *endhostent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17007 **NAME**

17008 gethostid — get an identifier for the current host

17009 **SYNOPSIS**

17010 XSI
```
#include <unistd.h>
```

17011
```
long gethostid(void);
```

17012

17013 **DESCRIPTION**

17014 The *gethostid*( ) function shall retrieve a 32-bit identifier for the current host.

17015 **RETURN VALUE**

17016 Upon successful completion, *gethostid*( ) shall return an identifier for the current host.

17017 **ERRORS**

17018 No errors are defined.

17019 **EXAMPLES**

17020 None.

17021 **APPLICATION USAGE**

17022 This volume of IEEE Std 1003.1-2001 does not define the domain in which the return value is
17023 unique.

17024 **RATIONALE**

17025 None.

17026 **FUTURE DIRECTIONS**

17027 None.

17028 **SEE ALSO**

17029 *random*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

17030 **CHANGE HISTORY**

17031 First released in Issue 4, Version 2.

17032 **Issue 5**

17033 Moved from X/OPEN UNIX extension to BASE.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17034 **NAME**

17035 gethostname — get name of current host

17036 **SYNOPSIS**

17037 `#include <unistd.h>`

17038 `int gethostname(char *`*`name`*`, size_t `*`namelen`*`);`

17039 **DESCRIPTION**

17040 The *gethostname*( ) function shall return the standard host name for the current machine. The
17041 *namelen* argument shall specify the size of the array pointed to by the *name* argument. The
17042 returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold
17043 the host name, then the returned name shall be truncated and it is unspecified whether the
17044 returned name is null-terminated.

17045 Host names are limited to {HOST_NAME_MAX} bytes.

17046 **RETURN VALUE**

17047 Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned.

17048 **ERRORS**

17049 No errors are defined.

17050 **EXAMPLES**

17051 None.

17052 **APPLICATION USAGE**

17053 None.

17054 **RATIONALE**

17055 None.

17056 **FUTURE DIRECTIONS**

17057 None.

17058 **SEE ALSO**

17059 *gethostid*( ), *uname*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

17060 **CHANGE HISTORY**

17061 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17062 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from
17063 **socklen_t** to **size_t**.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17064 **NAME**

17065 getitimer, setitimer — get and set value of interval timer

17066 **SYNOPSIS**

17067 XSI
```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *restrict value,
    struct itimerval *restrict ovalue);
```
17071

17072 **DESCRIPTION**

17073 The *getitimer*( ) function shall store the current value of the timer specified by *which* into the
17074 structure pointed to by *value*. The *setitimer*( ) function shall set the timer specified by *which* to
17075 the value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store
17076 the previous value of the timer in the structure pointed to by *ovalue*.

17077 A timer value is defined by the **itimerval** structure, specified in **<sys/time.h>**. If *it_value* is non-
17078 zero, it shall indicate the time to the next timer expiration. If *it_interval* is non-zero, it shall
17079 specify a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 shall
17080 disable a timer, regardless of the value of *it_interval*. Setting *it_interval* to 0 shall disable a timer
17081 after its next expiration (assuming *it_value* is non-zero).

17082 Implementations may place limitations on the granularity of timer values. For each interval
17083 timer, if the requested timer value requires a finer granularity than the implementation supports,
17084 the actual timer value shall be rounded up to the next supported value.

17085 An XSI-conforming implementation provides each process with at least three interval timers,
17086 which are indicated by the *which* argument:

17087 ITIMER_REAL Decrements in real time. A SIGALRM signal is delivered when this timer
17088 expires.

17089 ITIMER_VIRTUAL Decrements in process virtual time. It runs only when the process is
17090 executing. A SIGVTALRM signal is delivered when it expires.

17091 ITIMER_PROF Decrements both in process virtual time and when the system is running
17092 on behalf of the process. It is designed to be used by interpreters in
17093 statistically profiling the execution of interpreted programs. Each time the
17094 ITIMER_PROF timer expires, the SIGPROF signal is delivered.

17095 The interaction between *setitimer*( ) and any of *alarm*( ), *sleep*( ), or *usleep*( ) is unspecified.

17096 **RETURN VALUE**

17097 Upon successful completion, *getitimer*( ) or *setitimer*( ) shall return 0; otherwise, –1 shall be
17098 returned and *errno* set to indicate the error.

17099 **ERRORS**

17100 The *setitimer*( ) function shall fail if:

17101 [EINVAL] The *value* argument is not in canonical form. (In canonical form, the number of
17102 microseconds is a non-negative integer less than 1 000 000 and the number of
17103 seconds is a non-negative integer.)

17104 The *getitimer*( ) and *setitimer*( ) functions may fail if:

17105 [EINVAL] The *which* argument is not recognized.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17106 **EXAMPLES**

17107 None.

17108 **APPLICATION USAGE**

17109 None.

17110 **RATIONALE**

17111 None.

17112 **FUTURE DIRECTIONS**

17113 None.

17114 **SEE ALSO**

17115 *alarm*( ), *sleep*( ), *timer_getoverrun*( ), *ualarm*( ), *usleep*( ), the Base Definitions volume of
17116 IEEE Std 1003.1-2001, **<signal.h>**, **<sys/time.h>**

17117 **CHANGE HISTORY**

17118 First released in Issue 4, Version 2.

17119 **Issue 5**

17120 Moved from X/OPEN UNIX extension to BASE.

17121 **Issue 6**

17122 The **restrict** keyword is added to the *setitimer*( ) prototype for alignment with the
17123 ISO/IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

getlogin, getlogin_r — get login name

SYNOPSIS

```
#include <unistd.h>

char *getlogin(void);
int getlogin_r(char *name, size_t namesize);
```

DESCRIPTION

The *getlogin*( ) function shall return a pointer to a string containing the user name associated by the login activity with the controlling terminal of the current process. If *getlogin*( ) returns a non-null pointer, then that pointer points to the name that the user logged in under, even if there are several login names with the same user ID.

The *getlogin*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

TSF The *getlogin_r*( ) function shall put the name associated by the login activity with the controlling terminal of the current process in the character array pointed to by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum size of the login name is {LOGIN_NAME_MAX}.

If *getlogin_r*( ) is successful, *name* points to the name the user used at login, even if there are several login names with the same user ID.

RETURN VALUE

Upon successful completion, *getlogin*( ) shall return a pointer to the login name or a null pointer if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

The return value from *getlogin*( ) may point to static data whose content is overwritten by each call.

TSF If successful, the *getlogin_r*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *getlogin*( ) and *getlogin_r*( ) functions may fail if:

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENXIO] The calling process has no controlling terminal.

The *getlogin_r*( ) function may fail if:

TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Getting the User Login Name

The following example calls the *getlogin*( ) function to obtain the name of the user associated with the calling process, and passes this information to the *getpwnam*( ) function to get the associated user database information.

```
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
...
char *lgn;
struct passwd *pw;
...
if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
    fprintf(stderr, "Get of user information failed.\n"); exit(1);
    }
```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid*(*geteuid*( )) shall return the name associated with the effective user ID of the process; *getlogin*( ) shall return the name associated with the current login activity; and *getpwuid*(*getuid*( )) shall return the name associated with the real user ID of the process.

The *getlogin_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The *getlogin*( ) function returns a pointer to the user's login name. The same user ID may be shared by several login names. If it is desired to get the user database entry that is used during login, the result of *getlogin*( ) should be used to provide the argument to the *getpwnam*( ) function. (This might be used to determine the user's login shell, particularly where a single user has multiple login shells with distinct login names, but the same user ID.)

The information provided by the *cuserid*( ) function, which was originally defined in the POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
getpwuid(geteuid())
```

while the information provided by historical implementations of *cuserid*( ) can be obtained by:

```
getpwuid(getuid())
```

The thread-safe version of this function places the user name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwnam( ), *getpwuid*( ), *geteuid*( ), *getuid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<unistd.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17202 **CHANGE HISTORY**

17203 First released in Issue 1. Derived from System V Release 2.0.

17204 **Issue 5**

17205 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
17206 VALUE section.

17207 The *getlogin_r*( ) function is included for alignment with the POSIX Threads Extension.

17208 A note indicating that the *getlogin*( ) function need not be reentrant is added to the
17209 DESCRIPTION.

17210 **Issue 6**

17211 The *getlogin_r*( ) function is marked as part of the Thread-Safe Functions option.

17212 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

17213 The following new requirements on POSIX implementations derive from alignment with the
17214 Single UNIX Specification:

17215 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

17216 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

17217 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
17218 its avoidance of possibly using a static data area.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17219 **NAME**

17220 getmsg, getpmsg — receive next message from a STREAMS file (**STREAMS**)

17221 **SYNOPSIS**

17222 XSR
```
#include <stropts.h>

int getmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict flagsp);
int getpmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict bandp,
    int *restrict flagsp);
```

17228

17229 **DESCRIPTION**

17230 The *getmsg*( ) function shall retrieve the contents of a message located at the head of the
17231 STREAM head read queue associated with a STREAMS file and place the contents into one or
17232 more buffers. The message contains either a data part, a control part, or both. The data and
17233 control parts of the message shall be placed into separate buffers, as described below. The
17234 semantics of each part are defined by the originator of the message.

17235 The *getpmsg*( ) function shall be equivalent to *getmsg*( ), except that it provides finer control over
17236 the priority of the messages received. Except where noted, all requirements on *getmsg*( ) also
17237 pertain to *getpmsg*( ).

17238 The *fildes* argument specifies a file descriptor referencing a STREAMS-based file.

17239 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points
17240 to a buffer in which the data or control information is to be placed, and the *maxlen* member
17241 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall
17242 contain the number of bytes of data or control information actually received. The *len* member
17243 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to –1 if no data or
17244 control information is present in the message.

17245 When *getmsg*( ) is called, *flagsp* should point to an integer that indicates the type of message the
17246 process is able to receive. This is described further below.

17247 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold
17248 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is –1, the
17249 control (or data) part of the message shall not be processed and shall be left on the STREAM
17250 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to –1. If the
17251 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part
17252 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0
17253 and there are more than 0 bytes of control (or data) information, that information shall be left on
17254 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the
17255 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder
17256 of the message shall be left on the STREAM head read queue and a non-zero return value shall
17257 be provided.

17258 By default, *getmsg*( ) shall process the first available message on the STREAM head read queue.
17259 However, a process may choose to retrieve only high-priority messages by setting the integer
17260 pointed to by *flagsp* to RS_HIPRI. In this case, *getmsg*( ) shall only process the next message if it is
17261 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall
17262 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS_HIPRI if a
17263 high-priority message was retrieved, or 0 otherwise.

17264 For *getpmsg*( ), the flags are different. The *flagsp* argument points to a bitmask with the following
17265 mutually-exclusive flags defined: MSG_HIPRI, MSG_BAND, and MSG_ANY. Like *getmsg*( ),

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

getpmsg( ) shall process the first available message on the STREAM head read queue. A process may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to MSG_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg*( ) shall only process the next message if it is a high-priority message. In a similar manner, a process may choose to retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to MSG_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case, *getpmsg*( ) shall only process the next message if it is in a priority band equal to, or greater than, the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the first message off the queue, the integer pointed to by *flagsp* should be set to MSG_ANY and the integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-priority message, the integer pointed to by *flagsp* shall be set to MSG_HIPRI and the integer pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to MSG_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

If O_NONBLOCK is not set, *getmsg*( ) and *getpmsg*( ) shall block until a message of the type specified by *flagsp* is available at the front of the STREAM head read queue. If O_NONBLOCK is set and a message of the specified type is not present at the front of the read queue, *getmsg*( ) and *getpmsg*( ) shall fail and set *errno* to [EAGAIN].

If a hangup occurs on the STREAM from which messages are retrieved, *getmsg*( ) and *getpmsg*( ) shall continue to operate normally, as described above, until the STREAM head read queue is empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

RETURN VALUE

Upon successful completion, *getmsg*( ) and *getpmsg*( ) shall return a non-negative value. A value of 0 indicates that a full message was read successfully. A return value of MORECTL indicates that more control information is waiting for retrieval. A return value of MOREDATA indicates that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL and MOREDATA indicates that both types of information remain. Subsequent *getmsg*( ) and *getpmsg*( ) calls shall retrieve the remainder of the message. However, if a message of higher priority has come in on the STREAM head read queue, the next call to *getmsg*( ) or *getpmsg*( ) shall retrieve that higher-priority message before retrieving the remainder of the previous message.

If the high priority control part of the message is consumed, the message shall be placed back on the queue as a normal message of band 0. Subsequent *getmsg*( ) and *getpmsg*( ) calls shall retrieve the remainder of the message. If, however, a priority message arrives or already exists on the STREAM head, the subsequent call to *getmsg*( ) or *getpmsg*( ) shall retrieve the higher-priority message before retrieving the remainder of the message that was put back.

Upon failure, *getmsg*( ) and *getpmsg*( ) shall return –1 and set *errno* to indicate the error.

ERRORS

The *getmsg*( ) and *getpmsg*( ) functions shall fail if:

[EAGAIN] The O_NONBLOCK flag is set and no messages are available.

[EBADF] The *fildes* argument is not a valid file descriptor open for reading.

[EBADMSG] The queued message to be read is not valid for *getmsg*( ) or *getpmsg*( ) or a pending file descriptor is at the STREAM head.

[EINTR] A signal was caught during *getmsg*( ) or *getpmsg*( ).

[EINVAL] An illegal value was specified by *flagsp*, or the STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17312 [ENOSTR] A STREAM is not associated with *fildes*.

17313 In addition, *getmsg*( ) and *getpmsg*( ) shall fail if the STREAM head had processed an
17314 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
17315 *getmsg*( ) or *getpmsg*( ) but reflects the prior error.

17316 **EXAMPLES**

17317 **Getting Any Message**

17318 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call
17319 to *getmsg*( ) retrieves any available message on the associated STREAM-head read queue,
17320 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,
17321 respectively.

```
17322 #include <stropts.h>
17323 ...
17324 int fd;
17325 char ctrlbuf[128];
17326 char databuf[512];
17327 struct strbuf ctrl;
17328 struct strbuf data;
17329 int flags = 0;
17330 int ret;

17331 ctrl.buf = ctrlbuf;
17332 ctrl.maxlen = sizeof(ctrlbuf);

17333 data.buf = databuf;
17334 data.maxlen = sizeof(databuf);

17335 ret = getmsg (fd, &ctrl, &data, &flags);
```

17336 **Getting the First Message off the Queue**

17337 In the following example, the call to *getpmsg*( ) retrieves the first available message on the
17338 associated STREAM-head read queue.

```
17339 #include <stropts.h>
17340 ...

17341 int fd;
17342 char ctrlbuf[128];
17343 char databuf[512];
17344 struct strbuf ctrl;
17345 struct strbuf data;
17346 int band = 0;
17347 int flags = MSG_ANY;
17348 int ret;

17349 ctrl.buf = ctrlbuf;
17350 ctrl.maxlen = sizeof(ctrlbuf);

17351 data.buf = databuf;
17352 data.maxlen = sizeof(databuf);

17353 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17354 **APPLICATION USAGE**

17355 None.

17356 **RATIONALE**

17357 None.

17358 **FUTURE DIRECTIONS**

17359 None.

17360 **SEE ALSO**

17361 Section 2.6 (on page 38), *poll*( ), *putmsg*( ), *read*( ), *write*( ), the Base Definitions volume of
17362 IEEE Std 1003.1-2001, **<stropts.h>**

17363 **CHANGE HISTORY**

17364 First released in Issue 4, Version 2.

17365 **Issue 5**

17366 Moved from X/OPEN UNIX extension to BASE.

17367 A paragraph regarding ‘‘high-priority control parts of messages’’ is added to the RETURN
17368 VALUE section.

17369 **Issue 6**

17370 This function is marked as part of the XSI STREAMS Option Group.

17371 The **restrict** keyword is added to the *getmsg*( ) and *getpmsg*( ) prototypes for alignment with the
17372 ISO/IEC 9899:1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17373 **NAME**

17374 getnameinfo — get name information

17375 **SYNOPSIS**

17376 ```
#include <sys/socket.h>
```

17377 ```
#include <netdb.h>
```

17378 ```
int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
```

17379 ```
    char *restrict node, socklen_t nodelen, char *restrict service,
```

17380 ```
    socklen_t servicelen, int flags);
```

17381 **DESCRIPTION**

17382 The *getnameinfo*( ) function shall translate a socket address to a node name and service location,
17383 all of which are defined as in *getaddrinfo*( ).

17384 The *sa* argument points to a socket address structure to be translated.

17385 IP6 If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible
17386 IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node
17387 name for that IPv4 address.

17388 **Note:** The IPv6 unspecified address (`"::"`) and the IPv6 loopback address (`"::1"`) are not IPv4-
17389 compatible addresses. If the address is the IPv6 unspecified address (`"::"`), a lookup is not
17390 performed, and the [EAI_NONAME] error is returned.

17391 If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument
17392 points to a buffer able to contain up to *nodelen* characters that receives the node name as a null-
17393 terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name
17394 shall not be returned. If the node's name cannot be located, the numeric form of the address
17395 contained in the socket address structure pointed to by the *sa* argument is returned instead of its
17396 name.

17397 If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service*
17398 argument points to a buffer able to contain up to *servicelen* bytes that receives the service name
17399 as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero,
17400 the service name shall not be returned. If the service's name cannot be located, the numeric form
17401 of the service address (for example, its port number) shall be returned instead of its name.

17402 The *flags* argument is a flag that changes the default actions of the function. By default the fully-
17403 qualified domain name (FQDN) for the host shall be returned, but:

17404 • If the flag bit NI_NOFQDN is set, only the node name portion of the FQDN shall be returned
17405 for local hosts.

17406 • If the flag bit NI_NUMERICHOST is set, the numeric form of the address contained in the
17407 socket address structure pointed to by the *sa* argument shall be returned instead of its name,
17408 under all circumstances.

17409 • If the flag bit NI_NAMEREQD is set, an error shall be returned if the host's name cannot be
17410 located.

17411 • If the flag bit NI_NUMERICSERV is set, the numeric form of the service address shall be
17412 returned (for example, its port number) instead of its name, under all circumstances.

17413 • If the flag bit NI_NUMERICSCOPE is set, the numeric form of the scope identifier shall be
17414 returned (for example, interface index) instead of its name. This flag shall be ignored if the *sa*
17415 argument is not an IPv6 address.

17416 • If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service
17417 (SOCK_DGRAM). The default behavior shall assume that the service is a stream service


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
```

(SOCK_STREAM).

Notes:

1. The two NI_NUMERICxxx flags are required to support the **–n** flag that many commands provide.
2. The NI_DGRAM flag is required for the few AF_INET and AF_INET6 port numbers (for example, [512,514]) that represent different services for UDP and TCP.

The *getnameinfo*( ) function shall be thread-safe.

RETURN VALUE

A zero return value for *getnameinfo*( ) indicates successful completion; a non-zero return value indicates failure. The possible values for the failures are listed in the ERRORS section.

Upon successful completion, *getnameinfo*( ) shall return the *node* and *service* names, if requested, in the buffers provided. The returned names are always null-terminated strings.

ERRORS

The *getnameinfo*( ) function shall fail and return the corresponding value if:

[EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

[EAI_BADFLAGS]
The *flags* had an invalid value.

[EAI_FAIL] A non-recoverable error occurred.

[EAI_FAMILY] The address family was not recognized or the address length was invalid for the specified family.

[EAI_MEMORY] There was a memory allocation failure.

[EAI_NONAME] The name does not resolve for the supplied parameters.

NI_NAMEREQD is set and the host's name cannot be located, or both *nodename* and *servname* were null.

[EAI_OVERFLOW]
An argument buffer overflowed. The buffer pointed to by the *node* argument or the *service* argument was too small.

[EAI_SYSTEM] A system error occurred. The error code can be found in *errno*.

EXAMPLES

None.

APPLICATION USAGE

If the returned values are to be used as part of any further name resolution (for example, passed to *getaddrinfo*( )), applications should provide buffers large enough to store any result possible on the system.

Given the IPv4-mapped IPv6 address `"::ffff:1.2.3.4"`, the implementation performs a lookup as if the socket address structure contains the IPv4 address `"1.2.3.4"`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17458 **SEE ALSO**

17459 *gai_strerror*( ), *getaddrinfo*( ), *getservbyname*( ), *inet_ntop*( ), *socket*( ), the Base Definitions volume of
17460 IEEE Std 1003.1-2001, **<netdb.h>**, **<sys/socket.h>**

17461 **CHANGE HISTORY**

17462 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17463 The **restrict** keyword is added to the *getnameinfo*( ) prototype for alignment with the
17464 ISO/IEC 9899:1999 standard.

17465 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in |
17466 the SYNOPSIS and DESCRIPTION for alignment with IPv6. |

17467 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the |
17468 [EAI_OVERFLOW] error to the ERRORS section. |

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17469 **NAME**

17470 getnetbyaddr, getnetbyname, getnetent — network database functions

17471 **SYNOPSIS**

17472 `#include <netdb.h>`

17473 `struct netent *getnetbyaddr(uint32_t net, int type);`
17474 `struct netent *getnetbyname(const char *name);`
17475 `struct netent *getnetent(void);`

17476 **DESCRIPTION**

17477 Refer to *endnetent*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17478 **NAME**

17479 getopt, optarg, opterr, optind, optopt — command option parsing

17480 **SYNOPSIS**

17481 ```
#include <unistd.h>
```

17482 ```
int getopt(int argc, char * const argv[], const char *optstring);
```
17483 ```
extern char *optarg;
```
17484 ```
extern int optind, opterr, optopt;
```

17485 **DESCRIPTION**

17486 The *getopt*( ) function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,
17487 6, 7, 9, and 10 in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax
17488 Guidelines.

17489 The parameters *argc* and *argv* are the argument count and argument array as passed to *main*( )
17490 (see *exec*). The argument *optstring* is a string of recognized option characters; if a character is
17491 followed by a colon, the option takes an argument. All option characters allowed by Utility
17492 Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as
17493 an extension.

17494 The variable *optind* is the index of the next element of the *argv*[] vector to be processed. It shall
17495 be initialized to 1 by the system, and *getopt*( ) shall update it when it finishes with each element
17496 of *argv*[ ]. When an element of *argv*[ ] contains multiple option characters, it is unspecified how
17497 *getopt*( ) determines which options have already been processed.

17498 The *getopt*( ) function shall return the next option character (if one is found) from *argv* that
17499 matches a character in *optstring*, if there is one that matches. If the option takes an argument,
17500 *getopt*( ) shall set the variable *optarg* to point to the option-argument as follows:

17501 1. If the option was the last character in the string pointed to by an element of *argv*, then
17502 *optarg* shall contain the next element of *argv*, and *optind* shall be incremented by 2. If the
17503 resulting value of *optind* is greater than *argc*, this indicates a missing option-argument, and
17504 *getopt*( ) shall return an error indication.

17505 2. Otherwise, *optarg* shall point to the string following the option character in that element of
17506 *argv*, and *optind* shall be incremented by 1.

17507 If, when *getopt*( ) is called:

17508 ```
 argv[optind]   is a null pointer
17509 *argv[optind]   is not the character –
17510 argv[optind]   points to the string "–"
```

17511 *getopt*( ) shall return –1 without changing *optind*. If:

17512 ```
argv[optind]    points to the string "––"
```

17513 *getopt*( ) shall return –1 after incrementing *optind*.

17514 If *getopt*( ) encounters an option character that is not contained in *optstring*, it shall return the
17515 question-mark (`'?'`) character. If it detects a missing option-argument, it shall return the colon
17516 character (`':'`) if the first character of *optstring* was a colon, or a question-mark character (`'?'`)
17517 otherwise. In either case, *getopt*( ) shall set the variable *optopt* to the option character that caused
17518 the error. If the application has not set the variable *opterr* to 0 and the first character of *optstring*
17519 is not a colon, *getopt*( ) shall also print a diagnostic message to *stderr* in the format specified for
17520 the *getopts* utility.

17521 The *getopt*( ) function need not be reentrant. A function that is not required to be reentrant is not
17522 required to be thread-safe.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17523 **RETURN VALUE**

17524 The *getopt*( ) function shall return the next option character specified on the command line.

17525 A colon (':') shall be returned if *getopt*( ) detects a missing argument and the first character of
17526 *optstring* was a colon (':').

17527 A question mark ('?') shall be returned if *getopt*( ) encounters an option character not in
17528 *optstring* or detects a missing argument and the first character of *optstring* was not a colon (':').

17529 Otherwise, *getopt*( ) shall return –1 when all command line options are parsed.

17530 **ERRORS**

17531 No errors are defined.

17532 **EXAMPLES**

17533 **Parsing Command Line Options**

17534 The following code fragment shows how you might process the arguments for a utility that can
17535 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require
17536 arguments:

```
17537 #include <unistd.h>

17538 int
17539 main(int argc, char *argv[ ])
17540 {
17541     int c;
17542     int bflg, aflg, errflg;
17543     char *ifile;
17544     char *ofile;
17545     extern char *optarg;
17546     extern int optind, optopt;
17547     . . .
17548     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
17549         switch(c) {
17550         case 'a':
17551             if (bflg)
17552                 errflg++;
17553             else
17554                 aflg++;
17555             break;
17556         case 'b':
17557             if (aflg)
17558                 errflg++;
17559             else {
17560                 bflg++;
17561                 bproc();
17562             }
17563             break;
17564         case 'f':
17565             ifile = optarg;
17566             break;
17567         case 'o':
17568             ofile = optarg;
17569             break;
```


--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
                case ':':       /* -f or -o without operand */
                        fprintf(stderr,
                                "Option -%c requires an operand\n", optopt);
                        errflg++;
                        break;
            case '?':
                        fprintf(stderr,
                                "Unrecognized option: -%c\n", optopt);
                errflg++;
            }
        }
        if (errflg) {
            fprintf(stderr, "usage: . . . ");
            exit(2);
        }
        for ( ; optind < argc; optind++) {
            if (access(argv[optind], R_OK)) {
        . . .
    }
```

This code accepts any of the following as equivalent:

```
cmd –ao arg path path
cmd –a –o arg path path
cmd –o arg –a path path
cmd –a –o arg –– path path
cmd –a –oarg path path
cmd –aoarg path path
```

Checking Options and Arguments

The following example parses a set of command line options and prints messages to standard output for each option and argument that it encounters.

```
#include <unistd.h>
#include <stdio.h>
...
int c;
char *filename;
extern char *optarg;
extern int optind, optopt, opterr;
...
while ((c = getopt(argc, argv, ":abf:")) != -1) {
    switch(c) {
    case 'a':
        printf("a is set\n");
        break;
    case 'b':
        printf("b is set\n");
        break;
    case 'f':
        filename = optarg;
        printf("filename is %s\n", filename);
        break;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            case ':':
                printf("-%c without filename\n", optopt);
                break;
            case '?':
                printf("unknown arg %c\n", optopt);
                break;
            }
        }
```

Selecting Options from the Command Line

The following example selects the type of database routines the user wants to use based on the *Options* argument.

```
#include <unistd.h>
#include <string.h>
...
char *Options = "hdbtl";
...
int dbtype, i;
char c;
char *st;
...
dbtype = 0;
while ((c = getopt(argc, argv, Options)) != -1) {
    if ((st = strchr(Options, c)) != NULL) {
        dbtype = st - Options;
        break;
    }
}
```

APPLICATION USAGE

The *getopt*( ) function is only required to support option characters included in Utility Syntax Guideline 3. Many historical implementations of *getopt*( ) support other characters as options. This is an allowed extension, but applications that use extensions are not maximally portable. Note that support for multi-byte option characters is only possible when such characters can be represented as type **int**.

RATIONALE

The *optopt* variable represents historical practice and allows the application to obtain the identity of the invalid option.

The description has been written to make it clear that *getopt*( ), like the *getopts* utility, deals with option-arguments whether separated from the option by <blank>s or not. Note that the requirements on *getopt*( ) and *getopts* are more stringent than the Utility Syntax Guidelines.

The *getopt*( ) function shall return –1, rather than EOF, so that **<stdio.h>** is not required.

The special significance of a colon as the first character of *optstring* makes *getopt*( ) consistent with the *getopts* utility. It allows an application to make a distinction between a missing argument and an incorrect option letter without having to examine the option letter. It is true that a missing argument can only be detected in one case, but that is a case that has to be considered.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

exec, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *getopt*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

IEEE PASC Interpretation 1003.2 #150 is applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17675 **NAME**

17676 getpeername — get the name of the peer socket

17677 **SYNOPSIS**

17678
```
#include <sys/socket.h>
```

17679
```
int getpeername(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

17681 **DESCRIPTION**

17682 The *getpeername*( ) function shall retrieve the peer address of the specified socket, store this
17683 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this
17684 address in the object pointed to by the *address_len* argument.

17685 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
17686 the stored address shall be truncated.

17687 If the protocol permits connections by unbound clients, and the peer is not bound, then the value
17688 stored in the object pointed to by *address* is unspecified.

17689 **RETURN VALUE**

17690 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
17691 indicate the error.

17692 **ERRORS**

17693 The *getpeername*( ) function shall fail if:

17694 [EBADF] The *socket* argument is not a valid file descriptor.

17695 [EINVAL] The socket has been shut down.

17696 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

17697 [ENOTSOCK] The *socket* argument does not refer to a socket.

17698 [EOPNOTSUPP] The operation is not supported for the socket protocol.

17699 The *getpeername*( ) function may fail if:

17700 [ENOBUFS] Insufficient resources were available in the system to complete the call.

17701 **EXAMPLES**

17702 None.

17703 **APPLICATION USAGE**

17704 None.

17705 **RATIONALE**

17706 None.

17707 **FUTURE DIRECTIONS**

17708 None.

17709 **SEE ALSO**

17710 *accept*( ), *bind*( ), *getsockname*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
17711 **<sys/socket.h>**

17712 **CHANGE HISTORY**

17713 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17714 The **restrict** keyword is added to the *getpeername*( ) prototype for alignment with the
17715 ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17716 **NAME**

17717 getpgid — get the process group ID for a process

17718 **SYNOPSIS**

17719 XSI
```
#include <unistd.h>

pid_t getpgid(pid_t pid);
```
17720

17721

17722 **DESCRIPTION**

17723 The *getpgid*( ) function shall return the process group ID of the process whose process ID is equal
17724 to *pid*. If *pid* is equal to 0, *getpgid*( ) shall return the process group ID of the calling process.

17725 **RETURN VALUE**

17726 Upon successful completion, *getpgid*( ) shall return a process group ID. Otherwise, it shall return
17727 (**pid_t**)–1 and set *errno* to indicate the error.

17728 **ERRORS**

17729 The *getpgid*( ) function shall fail if:

17730 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the
17731 calling process, and the implementation does not allow access to the process
17732 group ID of that process from the calling process.

17733 [ESRCH] There is no process with a process ID equal to *pid*.

17734 The *getpgid*( ) function may fail if:

17735 [EINVAL] The value of the *pid* argument is invalid.

17736 **EXAMPLES**

17737 None.

17738 **APPLICATION USAGE**

17739 None.

17740 **RATIONALE**

17741 None.

17742 **FUTURE DIRECTIONS**

17743 None.

17744 **SEE ALSO**

17745 *exec*, *fork*( ), *getpgrp*( ), *getpid*( ), *getsid*( ), *setpgid*( ), *setsid*( ), the Base Definitions volume of
17746 IEEE Std 1003.1-2001, **<unistd.h>**

17747 **CHANGE HISTORY**

17748 First released in Issue 4, Version 2.

17749 **Issue 5**

17750 Moved from X/OPEN UNIX extension to BASE.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17751 **NAME**

17752 getpgrp — get the process group ID of the calling process

17753 **SYNOPSIS**

17754
```
#include <unistd.h>
```

17755
```
pid_t getpgrp(void);
```

17756 **DESCRIPTION**

17757 The *getpgrp*( ) function shall return the process group ID of the calling process.

17758 **RETURN VALUE**

17759 The *getpgrp*( ) function shall always be successful and no return value is reserved to indicate an
17760 error.

17761 **ERRORS**

17762 No errors are defined.

17763 **EXAMPLES**

17764 None.

17765 **APPLICATION USAGE**

17766 None.

17767 **RATIONALE**

17768 4.3 BSD provides a *getpgrp*( ) function that returns the process group ID for a specified process.
17769 Although this function supports job control, all known job control shells always specify the
17770 calling process with this function. Thus, the simpler System V *getpgrp*( ) suffices, and the added
17771 complexity of the 4.3 BSD *getpgrp*( ) is provided by the XSI extension *getpgid*( ).

17772 **FUTURE DIRECTIONS**

17773 None.

17774 **SEE ALSO**

17775 *exec*, *fork*( ), *getpgid*( ), *getpid*( ), *getppid*( ), *kill*( ), *setpgid*( ), *setsid*( ), the Base Definitions volume of
17776 IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

17777 **CHANGE HISTORY**

17778 First released in Issue 1. Derived from Issue 1 of the SVID.

17779 **Issue 6**

17780 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

17781 The following new requirements on POSIX implementations derive from alignment with the
17782 Single UNIX Specification:

17783 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
17784 required for conforming implementations of previous POSIX specifications, it was not
17785 required for UNIX applications.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17786 **NAME**

17787 getpid — get the process ID

17788 **SYNOPSIS**

17789
```
#include <unistd.h>
```

17790
```
pid_t getpid(void);
```

17791 **DESCRIPTION**

17792 The *getpid*( ) function shall return the process ID of the calling process.

17793 **RETURN VALUE**

17794 The *getpid*( ) function shall always be successful and no return value is reserved to indicate an
17795 error.

17796 **ERRORS**

17797 No errors are defined.

17798 **EXAMPLES**

17799 None.

17800 **APPLICATION USAGE**

17801 None.

17802 **RATIONALE**

17803 None.

17804 **FUTURE DIRECTIONS**

17805 None.

17806 **SEE ALSO**

17807 *exec*, *fork*( ), *getpgrp*( ), *getppid*( ), *kill*( ), *setpgid*( ), *setsid*( ), the Base Definitions volume of
17808 IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

17809 **CHANGE HISTORY**

17810 First released in Issue 1. Derived from Issue 1 of the SVID.

17811 **Issue 6**

17812 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

17813 The following new requirements on POSIX implementations derive from alignment with the
17814 Single UNIX Specification:

17815 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
17816 required for conforming implementations of previous POSIX specifications, it was not
17817 required for UNIX applications.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17818 **NAME**

17819 getpmsg — receive next message from a STREAMS file

17820 **SYNOPSIS**

17821 XSI
```
#include <stropts.h>

int getpmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict bandp,
    int *restrict flagsp);
```
17822

17823

17824

17825

17826 **DESCRIPTION**

17827 Refer to *getmsg*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

17828 **NAME**

17829 getppid — get the parent process ID

17830 **SYNOPSIS**

17831 `#include <unistd.h>`

17832 `pid_t getppid(void);`

17833 **DESCRIPTION**

17834 The *getppid*( ) function shall return the parent process ID of the calling process.

17835 **RETURN VALUE**

17836 The *getppid*( ) function shall always be successful and no return value is reserved to indicate an
17837 error.

17838 **ERRORS**

17839 No errors are defined.

17840 **EXAMPLES**

17841 None.

17842 **APPLICATION USAGE**

17843 None.

17844 **RATIONALE**

17845 None.

17846 **FUTURE DIRECTIONS**

17847 None.

17848 **SEE ALSO**

17849 *exec*, *fork*( ), *getpgid*( ), *getpgrp*( ), *getpid*( ), *kill*( ), *setpgid*( ), *setsid*( ), the Base Definitions volume of
17850 IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

17851 **CHANGE HISTORY**

17852 First released in Issue 1. Derived from Issue 1 of the SVID.

17853 **Issue 6**

17854 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

17855 The following new requirements on POSIX implementations derive from alignment with the
17856 Single UNIX Specification:

17857 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
17858 required for conforming implementations of previous POSIX specifications, it was not
17859 required for UNIX applications.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

getpriority, setpriority — get and set the nice value

SYNOPSIS

XSI
```
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int value);
```

DESCRIPTION

The *getpriority*( ) function shall obtain the nice value of a process, process group, or user. The *setpriority*( ) function shall set the nice value of a process, process group, or user to *value*+{NZERO}.

Target processes are specified by the values of the *which* and *who* arguments. The *which* argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an effective user ID, respectively. A 0 value for the *who* argument specifies the current process, process group, or user.

The nice value set with *setpriority*( ) shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

If more than one process is specified, *getpriority*( ) shall return value {NZERO} less than the lowest nice value pertaining to any of the specified processes, and *setpriority*( ) shall set the nice values of all of the specified processes to *value*+{NZERO}.

The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling. While the range of valid nice values is [0,{NZERO}*2–1], implementations may enforce more restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value, *setpriority*( ) shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater than the system's highest supported nice value, *setpriority*( ) shall set the nice value to the highest supported value.

Only a process with appropriate privileges can lower its nice value.

PS | TPS
Any processes or threads using SCHED_FIFO or SCHED_RR shall be unaffected by a call to *setpriority*( ). This is not considered an error. A process which subsequently reverts to SCHED_OTHER need not have its priority affected by such a *setpriority*( ) call.

The effect of changing the nice value may vary depending on the process-scheduling algorithm in effect.

Since *getpriority*( ) can return the value –1 on successful completion, it is necessary to set *errno* to 0 prior to a call to *getpriority*( ). If *getpriority*( ) returns the value –1, then *errno* can be checked to see if an error occurred or if the value is a legitimate nice value.

RETURN VALUE

Upon successful completion, *getpriority*( ) shall return an integer in the range –{NZERO} to {NZERO}–1. Otherwise, –1 shall be returned and *errno* set to indicate the error.

Upon successful completion, *setpriority*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getpriority*( ) and *setpriority*( ) functions shall fail if:

[ESRCH] No process could be located using the *which* and *who* argument values specified.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17905 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*
17906 argument is not a valid process ID, process group ID, or user ID.

17907 In addition, *setpriority*( ) may fail if:

17908 [EPERM] A process was located, but neither the real nor effective user ID of the
17909 executing process match the effective user ID of the process whose nice value
17910 is being changed.

17911 [EACCES] A request was made to change the nice value to a lower numeric value and
17912 the current process does not have appropriate privileges.

17913 **EXAMPLES**

17914 **Using getpriority( )**

17915 The following example returns the current scheduling priority for the process ID returned by the
17916 call to *getpid*( ).

```
17917 #include <sys/resource.h>
17918 ...
17919 int which = PRIO_PROCESS;
17920 id_t pid;
17921 int ret;

17922 pid = getpid();
17923 ret = getpriority(which, pid);
```

17924 **Using setpriority( )**

17925 The following example sets the priority for the current process ID to –20.

```
17926 #include <sys/resource.h>
17927 ...
17928 int which = PRIO_PROCESS;
17929 id_t pid;
17930 int priority = -20;
17931 int ret;

17932 pid = getpid();
17933 ret = setpriority(which, pid, priority);
```

17934 **APPLICATION USAGE**

17935 The *getpriority*( ) and *setpriority*( ) functions work with an offset nice value (nice value
17936 –{NZERO}). The nice value is in the range [0,2*{NZERO} –1], while the return value for
17937 *getpriority*( ) and the third parameter for *setpriority*( ) are in the range [–{NZERO},{NZERO} –1].

17938 **RATIONALE**

17939 None.

17940 **FUTURE DIRECTIONS**

17941 None.

17942 **SEE ALSO**

17943 *nice*( ), *sched_get_priority_max*( ), *sched_setscheduler*( ), the Base Definitions volume of
17944 IEEE Std 1003.1-2001, **<sys/resource.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

17945 **CHANGE HISTORY**

17946 First released in Issue 4, Version 2.

17947 **Issue 5**

17948 Moved from X/OPEN UNIX extension to BASE.

17949 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion
17950 with functionality in the POSIX Realtime Extension.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

17951 **NAME**

17952 getprotobyname, getprotobynumber, getprotent — network protocol database functions

17953 **SYNOPSIS**

17954 `#include <netdb.h>`

17955 `struct protoent *getprotobyname(const char *`*`name`*`);`
17956 `struct protoent *getprotobynumber(int` *`proto`*`);`
17957 `struct protoent *getprotoent(void);`

17958 **DESCRIPTION**

17959 Refer to *endprotoent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`-`,,`,`,`,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

17960 **NAME**

17961 getpwent — get user database entry

17962 **SYNOPSIS**

17963 XSI
```
#include <pwd.h>
```

17964
```
struct passwd *getpwent(void);
```

17965

17966 **DESCRIPTION**

17967 Refer to *endpwent*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

17968 **NAME**

17969 getpwnam, getpwnam_r — search user database for a name

17970 **SYNOPSIS**

17971 ```
#include <pwd.h>
```

17972 ```
struct passwd *getpwnam(const char *name);
```

17973 TSF ```
int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
```

17974 ```
    size_t bufsize, struct passwd **result);
```

17975

17976 **DESCRIPTION**

17977 The *getpwnam*( ) function shall search the user database for an entry with a matching *name*.

17978 The *getpwnam*( ) function need not be reentrant. A function that is not required to be reentrant is
17979 not required to be thread-safe.

17980 Applications wishing to check for error situations should set *errno* to 0 before calling
17981 *getpwnam*( ). If *getpwnam*( ) returns a null pointer and *errno* is non-zero, an error occurred.

17982 TSF The *getpwnam_r*( ) function shall update the **passwd** structure pointed to by *pwd* and store a
17983 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
17984 from the user database with a matching *name*. Storage referenced by the structure is allocated
17985 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The
17986 maximum size needed for this buffer can be determined with the {_SC_GETPW_R_SIZE_MAX}
17987 *sysconf*( ) parameter. A NULL pointer shall be returned at the location pointed to by *result* on
17988 error or if the requested entry is not found.

17989 **RETURN VALUE**

17990 The *getpwnam*( ) function shall return a pointer to a **struct passwd** with the structure as defined
17991 in **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested
17992 entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

17993 The return value may point to a static area which is overwritten by a subsequent call to
17994 *getpwent*( ), *getpwnam*( ), or *getpwuid*( ).

17995 TSF If successful, the *getpwnam_r*( ) function shall return zero; otherwise, an error number shall be
17996 returned to indicate the error.

17997 **ERRORS**

17998 The *getpwnam*( ) and *getpwnam_r*( ) functions may fail if:

17999 [EIO] An I/O error has occurred.

18000 [EINTR] A signal was caught during *getpwnam*( ).

18001 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

18002 [ENFILE] The maximum allowable number of files is currently open in the system.

18003 The *getpwnam_r*( ) function may fail if:

18004 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to
18005 be referenced by the resulting **passwd** structure.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Getting an Entry for the Login Name

The following example uses the *getlogin*( ) function to return the name of the user who logged in; this information is passed to the *getpwnam*( ) function to get the user database entry for that user.

```
#include <sys/types.h>
#include <pwd.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
...
char *lgn;
struct passwd *pw;
...
if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
    fprintf(stderr, "Get of user information failed.\n"); exit(1);
}
...
```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid*(*geteuid*( )) returns the name associated with the effective user ID of the process; *getlogin*( ) returns the name associated with the current login activity; and *getpwuid*(*getuid*( )) returns the name associated with the real user ID of the process.

The *getpwnam_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwuid( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<pwd.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getpwnam_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getpwnam*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getpwnam_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *name*.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

getpwuid, getpwuid_r — search user database for a user ID

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
TSF int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
        size_t bufsize, struct passwd **result);
```

DESCRIPTION

The *getpwuid*( ) function shall search the user database for an entry with a matching *uid*.

The *getpwuid*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid*( ). If *getpwuid*( ) returns a null pointer and *errno* is set to non-zero, an error occurred.

TSF The *getpwuid_r*( ) function shall update the **passwd** structure pointed to by *pwd* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the user database with a matching *uid*. Storage referenced by the structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The maximum size needed for this buffer can be determined with the {_SC_GETPW_R_SIZE_MAX} *sysconf*( ) parameter. A NULL pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

The *getpwuid*( ) function shall return a pointer to a **struct passwd** with the structure as defined in **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

The return value may point to a static area which is overwritten by a subsequent call to *getpwent*( ), *getpwnam*( ), or *getpwuid*( ).

TSF If successful, the *getpwuid_r*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *getpwuid*( ) and *getpwuid_r*( ) functions may fail if:

[EIO]	An I/O error has occurred.
[EINTR]	A signal was caught during *getpwuid*( ).
[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
[ENFILE]	The maximum allowable number of files is currently open in the system.

The *getpwuid_r*( ) function may fail if:

TSF [ERANGE]	Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **passwd** structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18100 **EXAMPLES**

18101 **Getting an Entry for the Root User**

18102 The following example gets the user database entry for the user with user ID 0 (root).

```
18103 #include <sys/types.h>
18104 #include <pwd.h>
18105 ...
18106 uid_t id = 0;
18107 struct passwd *pwd;
18108 pwd = getpwuid(id);
```

18109 **Finding the Name for the Effective User ID**

18110 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to
18111 store the structure pointer returned by the call to the *getpwuid*( ) function. The *geteuid*( ) function
18112 shall return the effective user ID of the calling process; this is used as the search criteria for the
18113 *getpwuid*( ) function. The call to *getpwuid*( ) shall return a pointer to the structure containing that
18114 user ID value.

```
18115 #include <unistd.h>
18116 #include <sys/types.h>
18117 #include <pwd.h>
18118 ...
18119 struct passwd *pws;
18120 pws = getpwuid(geteuid());
```

18121 **Finding an Entry in the User Database**

18122 The following example uses *getpwuid*( ) to search the user database for a user ID that was
18123 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not
18124 found, the program prints the numeric value of the user ID for the entry.

```
18125 #include <sys/types.h>
18126 #include <pwd.h>
18127 #include <stdio.h>
18128 ...
18129 struct stat statbuf;
18130 struct passwd *pwd;
18131 ...
18132 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
18133     printf(" %-8.8s", pwd->pw_name);
18134 else
18135     printf(" %-8d", statbuf.st_uid);
```

18136 **APPLICATION USAGE**

18137 Three names associated with the current process can be determined: *getpwuid*(*geteuid*( )) returns
18138 the name associated with the effective user ID of the process; *getlogin*( ) returns the name
18139 associated with the current login activity; and *getpwuid*(*getuid*( )) returns the name associated
18140 with the real user ID of the process.

18141 The *getpwuid_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of
18142 possibly using a static data area that may be overwritten by each call.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwnam( ), *geteuid*( ), *getuid*( ), *getlogin*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<pwd.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getpwuid_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getpwuid*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getpwuid_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *uid*.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

getrlimit, setrlimit — control maximum resource consumption

SYNOPSIS

XSI
```
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);
```

DESCRIPTION

The *getrlimit*( ) function shall get, and the *setrlimit*( ) function shall set, limits on the consumption of a variety of resources.

Each call to either *getrlimit*( ) or *setrlimit*( ) identifies a specific resource to be operated upon as well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim_cur* member specifies the current or soft limit and the *rlim_max* member specifies the maximum or hard limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both hard and soft limits can be changed in a single call to *setrlimit*( ) subject to the constraints described above.

The value RLIM_INFINITY, defined in **<sys/resource.h>**, shall be considered to be larger than any other limit value. If a call to *getrlimit*( ) returns RLIM_INFINITY for a resource, it means the implementation shall not enforce limits on that resource. Specifying RLIM_INFINITY as any resource limit value on a successful call to *setrlimit*( ) shall inhibit enforcement of that resource limit.

The following resources are defined:

RLIMIT_CORE
: This is the maximum size of a **core** file, in bytes, that may be created by a process. A limit of 0 shall prevent the creation of a **core** file. If this limit is exceeded, the writing of a **core** file shall terminate at this size.

RLIMIT_CPU
: This is the maximum amount of CPU time, in seconds, used by a process. If this limit is exceeded, SIGXCPU shall be generated for the process. If the process is catching or ignoring SIGXCPU, or all threads belonging to that process are blocking SIGXCPU, the behavior is unspecified.

RLIMIT_DATA
: This is the maximum size of a process' data segment, in bytes. If this limit is exceeded, the *malloc*( ) function shall fail with *errno* set to [ENOMEM].

RLIMIT_FSIZE
: This is the maximum size of a file, in bytes, that may be created by a process. If a write or truncate operation would cause this limit to be exceeded, SIGXFSZ shall be generated for the thread. If the thread is blocking, or the process is catching or ignoring SIGXFSZ, continued attempts to increase the size of a file from end-of-file to beyond the limit shall fail with *errno* set to [EFBIG].

RLIMIT_NOFILE
: This is a number one greater than the maximum value that the system may assign to a newly-created descriptor. If this limit is exceeded, functions that allocate a file descriptor shall fail with *errno* set to [EMFILE]. This limit constrains the number of file descriptors that a process may allocate.

RLIMIT_STACK
: This is the maximum size of a process' stack, in bytes. The implementation does not automatically grow the stack beyond this limit.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18221 If this limit is exceeded, SIGSEGV shall be generated for the thread. If the
18222 thread is blocking SIGSEGV, or the process is ignoring or catching
18223 SIGSEGV and has not made arrangements to use an alternate stack, the
18224 disposition of SIGSEGV shall be set to SIG_DFL before it is generated.

18225 RLIMIT_AS This is the maximum size of a process' total available memory, in bytes. If
18226 this limit is exceeded, the *malloc*( ) and *mmap*( ) functions shall fail with
18227 *errno* set to [ENOMEM]. In addition, the automatic stack growth fails
18228 with the effects outlined above.

18229 When using the *getrlimit*( ) function, if a resource limit can be represented correctly in an object
18230 of type **rlim_t**, then its representation is returned; otherwise, if the value of the resource limit is
18231 equal to that of the corresponding saved hard limit, the value returned shall be
18232 RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.

18233 When using the *setrlimit*( ) function, if the requested new limit is RLIM_INFINITY, the new limit
18234 shall be ''no limit''; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit
18235 shall be the corresponding saved hard limit; otherwise, if the requested new limit is
18236 RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the
18237 new limit shall be the requested value. In addition, if the corresponding saved limit can be
18238 represented correctly in an object of type **rlim_t** then it shall be overwritten with the new limit.

18239 The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless
18240 a previous call to *getrlimit*( ) returned that value as the soft or hard limit for the corresponding
18241 resource limit.

18242 The determination of whether a limit can be correctly represented in an object of type **rlim_t** is
18243 implementation-defined. For example, some implementations permit a limit whose value is
18244 greater than RLIM_INFINITY and others do not.

18245 The *exec* family of functions shall cause resource limits to be saved.

18246 **RETURN VALUE**

18247 Upon successful completion, *getrlimit*( ) and *setrlimit*( ) shall return 0. Otherwise, these functions
18248 shall return –1 and set *errno* to indicate the error.

18249 **ERRORS**

18250 The *getrlimit*( ) and *setrlimit*( ) functions shall fail if:

18251 [EINVAL] An invalid *resource* was specified; or in a *setrlimit*( ) call, the new *rlim_cur*
18252 exceeds the new *rlim_max*.

18253 [EPERM] The limit specified to *setrlimit*( ) would have raised the maximum limit value,
18254 and the calling process does not have appropriate privileges.

18255 The *setrlimit*( ) function may fail if:

18256 [EINVAL] The limit specified cannot be lowered because current usage is already higher
18257 than the limit.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the value of {_POSIX_OPEN_MAX} from **<limits.h>**, unexpected behavior may occur.

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the highest currently open file descriptor +1, unexpected behavior may occur.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *malloc*( ), *open*( ), *sigaltstack*( ), *sysconf*( ), *ulimit*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stropts.h>**, **<sys/resource.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

An APPLICATION USAGE section is added.

Large File Summit extensions are added.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for RLIMIT_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of a process attempting to set the hard or soft limit for RLIMIT_NOFILE less than the highest currently open file descriptor +1.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

getrusage — get information about resource utilization

SYNOPSIS

XSI

```
#include <sys/resource.h>

int getrusage(int who, struct rusage *r_usage);
```

DESCRIPTION

The *getrusage*( ) function shall provide measures of the resources used by the current process or its terminated and waited-for child processes. If the value of the *who* argument is RUSAGE_SELF, information shall be returned about resources used by the current process. If the value of the *who* argument is RUSAGE_CHILDREN, information shall be returned about resources used by the terminated and waited-for children of the current process. If the child is never waited for (for example, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to SIG_IGN), the resource information for the child process is discarded and not included in the resource information provided by *getrusage*( ).

The *r_usage* argument is a pointer to an object of type **struct rusage** in which the returned information is stored.

RETURN VALUE

Upon successful completion, *getrusage*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getrusage*( ) function shall fail if:

[EINVAL] The value of the *who* argument is not valid.

EXAMPLES

Using getrusage( )

The following example returns information about the resources used by the current process.

```
#include <sys/resource.h>
...
int who = RUSAGE_SELF;
struct rusage usage;
int ret;

ret = getrusage(who, &usage);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *sigaction*( ), *time*( ), *times*( ), *wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/resource.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18325 **CHANGE HISTORY**

18326 First released in Issue 4, Version 2.

18327 **Issue 5**

18328 Moved from X/OPEN UNIX extension to BASE.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18329 **NAME**

18330 gets — get a string from a stdin stream

18331 **SYNOPSIS**

18332 `#include <stdio.h>`

18333 `char *gets(char *s);`

18334 **DESCRIPTION**

18335 CX The functionality described on this reference page is aligned with the ISO C standard. Any
18336 conflict between the requirements described here and the ISO C standard is unintentional. This
18337 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18338 The *gets*( ) function shall read bytes from the standard input stream, *stdin*, into the array pointed
18339 to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall
18340 be discarded and a null byte shall be placed immediately after the last byte read into the array.

18341 CX The *gets*( ) function may mark the *st_atime* field of the file associated with *stream* for update. The
18342 *st_atime* field shall be marked for update by the first successful execution of *fgetc*( ), *fgets*( ),
18343 *fread*( ), *getc*( ), *getchar*( ), *gets*( ), *fscanf*( ), or *scanf*( ) using *stream* that returns data not supplied by
18344 a prior call to *ungetc*( ).

18345 **RETURN VALUE**

18346 Upon successful completion, *gets*( ) shall return *s*. If the stream is at end-of-file, the end-of-file
18347 indicator for the stream shall be set and *gets*( ) shall return a null pointer. If a read error occurs,
18348 CX the error indicator for the stream shall be set, *gets*( ) shall return a null pointer, and set *errno* to
18349 indicate the error.

18350 **ERRORS**

18351 Refer to *fgetc*( ).

18352 **EXAMPLES**

18353 None.

18354 **APPLICATION USAGE**

18355 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of
18356 *fgets*( ) is recommended.

18357 Since the user cannot specify the length of the buffer passed to *gets*( ), use of this function is
18358 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in
18359 such a way as to cause applications to fail, or possible system security violations.

18360 It is recommended that the *fgets*( ) function should be used to read input lines.

18361 **RATIONALE**

18362 None.

18363 **FUTURE DIRECTIONS**

18364 None.

18365 **SEE ALSO**

18366 *feof*( ), *ferror*( ), *fgets*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

18367 **CHANGE HISTORY**

18368 First released in Issue 1. Derived from Issue 1 of the SVID.

18369 **Issue 6**

18370 Extensions beyond the ISO C standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18371 **NAME**

18372 getservbyname, getservbyport, getservent — network services database functions

18373 **SYNOPSIS**

18374 `#include <netdb.h>`

18375 `struct servent *getservbyname(const char *`*`name`*`, const char *`*`proto`*`);`
18376 `struct servent *getservbyport(int` *`port`*`, const char *`*`proto`*`);`
18377 `struct servent *getservent(void);`

18378 **DESCRIPTION**

18379 Refer to *endservent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18380 **NAME**

18381 getsid — get the process group ID of a session leader

18382 **SYNOPSIS**

18383 XSI
```
#include <unistd.h>

pid_t getsid(pid_t pid);
```
18384

18385

18386 **DESCRIPTION**

18387 The *getsid*( ) function shall obtain the process group ID of the process that is the session leader of
18388 the process specified by *pid*. If *pid* is (**pid_t**)0, it specifies the calling process.

18389 **RETURN VALUE**

18390 Upon successful completion, *getsid*( ) shall return the process group ID of the session leader of
18391 the specified process. Otherwise, it shall return (**pid_t**)–1 and set *errno* to indicate the error.

18392 **ERRORS**

18393 The *getsid*( ) function shall fail if:

18394 [EPERM] The process specified by *pid* is not in the same session as the calling process,
18395 and the implementation does not allow access to the process group ID of the
18396 session leader of that process from the calling process.

18397 [ESRCH] There is no process with a process ID equal to *pid*.

18398 **EXAMPLES**

18399 None.

18400 **APPLICATION USAGE**

18401 None.

18402 **RATIONALE**

18403 None.

18404 **FUTURE DIRECTIONS**

18405 None.

18406 **SEE ALSO**

18407 *exec*, *fork*( ), *getpid*( ), *getpgid*( ), *setpgid*( ), *setsid*( ), the Base Definitions volume of
18408 IEEE Std 1003.1-2001, **<unistd.h>**

18409 **CHANGE HISTORY**

18410 First released in Issue 4, Version 2.

18411 **Issue 5**

18412 Moved from X/OPEN UNIX extension to BASE.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18413 **NAME**

18414 getsockname — get the socket name

18415 **SYNOPSIS**

18416 `#include <sys/socket.h>`

18417 `int getsockname(int` *`socket`*`, struct sockaddr *restrict` *`address`*`,`

18418 `socklen_t *restrict` *`address_len`*`);`

18419 **DESCRIPTION**

18420 The *getsockname*( ) function shall retrieve the locally-bound name of the specified socket, store
18421 this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of
18422 this address in the object pointed to by the *address_len* argument.

18423 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
18424 the stored address shall be truncated.

18425 If the socket has not been bound to a local name, the value stored in the object pointed to by
18426 *address* is unspecified.

18427 **RETURN VALUE**

18428 Upon successful completion, 0 shall be returned, the *address* argument shall point to the address
18429 of the socket, and the *address_len* argument shall point to the length of the address. Otherwise, –1
18430 shall be returned and *errno* set to indicate the error.

18431 **ERRORS**

18432 The *getsockname*( ) function shall fail if:

18433 [EBADF] The *socket* argument is not a valid file descriptor.

18434 [ENOTSOCK] The *socket* argument does not refer to a socket.

18435 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

18436 The *getsockname*( ) function may fail if:

18437 [EINVAL] The socket has been shut down.

18438 [ENOBUFS] Insufficient resources were available in the system to complete the function.

18439 **EXAMPLES**

18440 None.

18441 **APPLICATION USAGE**

18442 None.

18443 **RATIONALE**

18444 None.

18445 **FUTURE DIRECTIONS**

18446 None.

18447 **SEE ALSO**

18448 *accept*( ), *bind*( ), *getpeername*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
18449 **<sys/socket.h>**

18450 **CHANGE HISTORY**

18451 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18452 The **restrict** keyword is added to the *getsockname*( ) prototype for alignment with the
18453 ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

getsockopt — get the socket options

SYNOPSIS

```
#include <sys/socket.h>

int getsockopt(int socket, int level, int option_name,
    void *restrict option_value, socklen_t *restrict option_len);
```

DESCRIPTION

The *getsockopt*( ) function manipulates options associated with a socket.

The *getsockopt*( ) function shall retrieve the value for the option specified by the *option_name* argument for the socket specified by the *socket* argument. If the size of the option value is greater than *option_len*, the value stored in the object pointed to by the *option_value* argument shall be silently truncated. Otherwise, the object pointed to by the *option_len* argument shall be modified to indicate the actual length of the value.

The *level* argument specifies the protocol level at which the option resides. To retrieve options at the socket level, specify the *level* argument as SOL_SOCKET. To retrieve options at other levels, supply the appropriate level identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to IPPROTO_TCP as defined in the **<netinet/in.h>** header.

The socket in use may require the process to have appropriate privileges to use the *getsockopt*( ) function.

The *option_name* argument specifies a single option to be retrieved. It can be one of the following values defined in **<sys/socket.h>**:

SO_DEBUG	Reports whether debugging information is being recorded. This option shall store an **int** value. This is a Boolean option.
SO_ACCEPTCONN	Reports whether socket listening is enabled. This option shall store an **int** value. This is a Boolean option.
SO_BROADCAST	Reports whether transmission of broadcast messages is supported, if this is supported by the protocol. This option shall store an **int** value. This is a Boolean option.
SO_REUSEADDR	Reports whether the rules used in validating addresses supplied to *bind*( ) should allow reuse of local addresses, if this is supported by the protocol. This option shall store an **int** value. This is a Boolean option.
SO_KEEPALIVE	Reports whether connections are kept active with periodic transmission of messages, if this is supported by the protocol. If the connected socket fails to respond to these messages, the connection shall be broken and threads writing to that socket shall be notified with a SIGPIPE signal. This option shall store an **int** value. This is a Boolean option.
SO_LINGER	Reports whether the socket lingers on *close*( ) if data is present. If SO_LINGER is set, the system blocks the process during *close*( ) until it can transmit the data or until the end of the interval indicated by the *l_linger* member, whichever comes first. If SO_LINGER is not specified, and *close*( ) is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option shall store a **linger** structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SO_OOBINLINE Reports whether the socket leaves received out-of-band data (data marked urgent) inline. This option shall store an **int** value. This is a Boolean option.

SO_SNDBUF Reports send buffer size information. This option shall store an **int** value.

SO_RCVBUF Reports receive buffer size information. This option shall store an **int** value.

SO_ERROR Reports information about error status and clears it. This option shall store an **int** value.

SO_TYPE Reports the socket type. This option shall store an **int** value. Socket types are described in Section 2.10.6 (on page 59).

SO_DONTROUTE Reports whether outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option shall store an **int** value. This is a Boolean option.

SO_RCVLOWAT Reports the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option shall store an **int** value. Note that not all implementations allow this option to be retrieved.

SO_RCVTIMEO Reports the timeout value for input operations. This option shall store a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or *errno* set to [EAGAIN] or [EWOULDBLOCK] if no data was received. The default for this option is zero, which indicates that a receive operation shall not time out. Note that not all implementations allow this option to be retrieved.

SO_SNDLOWAT Reports the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option shall store an **int** value. Note that not all implementations allow this option to be retrieved.

SO_SNDTIMEO Reports the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with *errno* set to [EAGAIN] or [EWOULDBLOCK] if no data was sent. The default for this option is zero, which indicates that a send operation shall not time out. The option shall store a **timeval** structure. Note that not all implementations allow this option to be retrieved.

For Boolean options, a zero value indicates that the option is disabled and a non-zero value indicates that the option is enabled.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18546 **RETURN VALUE**

18547 Upon successful completion, *getsockopt*( ) shall return 0; otherwise, –1 shall be returned and *errno*
18548 set to indicate the error.

18549 **ERRORS**

18550 The *getsockopt*( ) function shall fail if:

18551 [EBADF] The *socket* argument is not a valid file descriptor.

18552 [EINVAL] The specified option is invalid at the specified socket level.

18553 [ENOPROTOOPT]
18554 The option is not supported by the protocol.

18555 [ENOTSOCK] The *socket* argument does not refer to a socket.

18556 The *getsockopt*( ) function may fail if:

18557 [EACCES] The calling process does not have the appropriate privileges.

18558 [EINVAL] The socket has been shut down.

18559 [ENOBUFS] Insufficient resources are available in the system to complete the function.

18560 **EXAMPLES**

18561 None.

18562 **APPLICATION USAGE**

18563 None.

18564 **RATIONALE**

18565 None.

18566 **FUTURE DIRECTIONS**

18567 None.

18568 **SEE ALSO**

18569 *bind*( ), *close*( ), *endprotoent*( ), *setsockopt*( ), *socket*( ), the Base Definitions volume of
18570 IEEE Std 1003.1-2001, **<sys/socket.h>**, **<netinet/in.h>**

18571 **CHANGE HISTORY**

18572 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18573 The **restrict** keyword is added to the *getsockopt*( ) prototype for alignment with the
18574 ISO / IEC 9899: 1999 standard.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18575 **NAME**

18576 getsubopt — parse suboption arguments from a string

18577 **SYNOPSIS**

18578 XSI
```
#include <stdlib.h>

int getsubopt(char **optionp, char * const *keylistp, char **valuep);
```
18579

18580

18581 **DESCRIPTION**

18582 The *getsubopt*( ) function shall parse suboption arguments in a flag argument. Such options often
18583 result from the use of *getopt*( ).

18584 The *getsubopt*( ) argument *optionp* is a pointer to a pointer to the option argument string. The
18585 suboption arguments shall be separated by commas and each may consist of either a single
18586 token, or a token-value pair separated by an equal sign.

18587 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified
18588 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in
18589 **optionp*. Since commas delimit suboption arguments in *optionp*, they should not appear in any of
18590 the strings pointed to by *keylistp*. Similarly, because an equal sign separates a token from its
18591 value, the application should not include an equal sign in any of the strings pointed to by
18592 *keylistp*.

18593 The *valuep* argument is the address of a value string pointer.

18594 If a comma appears in *optionp*, it shall be interpreted as a suboption separator. After commas
18595 have been processed, if there are one or more equal signs in a suboption string, the first equal
18596 sign in any suboption string shall be interpreted as a separator between a token and a value.
18597 Subsequent equal signs in a suboption string shall be interpreted as part of the value.

18598 If the string at **optionp* contains only one suboption argument (equivalently, no commas),
18599 *getsubopt*( ) shall update **optionp* to point to the null character at the end of the string. Otherwise,
18600 it shall isolate the suboption argument by replacing the comma separator with a null character,
18601 and shall update **optionp* to point to the start of the next suboption argument. If the suboption
18602 argument has an associated value (equivalently, contains an equal sign), *getsubopt*( ) shall update
18603 **valuep* to point to the value's first character. Otherwise, it shall set **valuep* to a null pointer. The
18604 calling application may use this information to determine whether the presence or absence of a
18605 value for the suboption is an error.

18606 Additionally, when *getsubopt*( ) fails to match the suboption argument with a token in the *keylistp*
18607 array, the calling application should decide if this is an error, or if the unrecognized option
18608 should be processed in another way.

18609 **RETURN VALUE**

18610 The *getsubopt*( ) function shall return the index of the matched token string, or –1 if no token
18611 strings were matched.

18612 **ERRORS**

18613 No errors are defined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

```
#include <stdio.h>
#include <stdlib.h>

int do_all;
const char *type;
int read_size;
int write_size;
int read_only;

enum
{
    RO_OPTION = 0,
    RW_OPTION,
    READ_SIZE_OPTION,
    WRITE_SIZE_OPTION
};

const char *mount_opts[] =
{
    [RO_OPTION] = "ro",
    [RW_OPTION] = "rw",
    [READ_SIZE_OPTION] = "rsize",
    [WRITE_SIZE_OPTION] = "wsize",
    NULL
};

int
main(int argc, char *argv[])
{
    char *subopts, *value;
    int opt;

    while ((opt = getopt(argc, argv, "at:o:")) != -1)
        switch(opt)
            {
            case 'a':
                do_all = 1;
                break;
            case 't':
                type = optarg;
                break;
            case 'o':
                subopts = optarg;
                while (*subopts != '\0')
                    switch(getsubopt(&subopts, mount_opts, &value))
                {
                case RO_OPTION:
                    read_only = 1;
                    break;
                case RW_OPTION:
                    read_only = 0;
                    break;
                case READ_SIZE_OPTION:
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
                        if (value == NULL)
                            abort();
                        read_size = atoi(value);
                        break;
                    case WRITE_SIZE_OPTION:
                        if (value == NULL)
                            abort();
                        write_size = atoi(value);
                        break;
                    default:
                        /* Unknown suboption. */
                        printf("Unknown suboption `%s'\n", value);
                        break;
                    }
                break;
            default:
                abort();
            }

        /* Do the real work. */

        return 0;
    }
```

Parsing Suboptions

The following example uses the *getsubopt*( ) function to parse a *value* argument in the *optarg* external variable returned by a call to *getopt*( ).

```
#include <stdlib.h>
...
char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
char *value;
int opt, index;

while ((opt = getopt(argc, argv, "e:")) != -1) {
    switch(opt)  {
    case 'e' :
        while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
            switch(index) {
...
            }
            break;
...
    }
}
...
```

APPLICATION USAGE

None.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18708 **FUTURE DIRECTIONS**

18709 None.

18710 **SEE ALSO**

18711 *getopt*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

18712 **CHANGE HISTORY**

18713 First released in Issue 4, Version 2.

18714 **Issue 5**

18715 Moved from X/OPEN UNIX extension to BASE.

18716 **Issue 6**

18717 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error
18718 in the SYNOPSIS.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18719 **NAME**

18720 gettimeofday — get the date and time

18721 **SYNOPSIS**

18722 XSI
```
#include <sys/time.h>

int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```
18723

18724

18725 **DESCRIPTION**

18726 The *gettimeofday*( ) function shall obtain the current time, expressed as seconds and
18727 microseconds since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The
18728 resolution of the system clock is unspecified.

18729 If *tzp* is not a null pointer, the behavior is unspecified.

18730 **RETURN VALUE**

18731 The *gettimeofday*( ) function shall return 0 and no value shall be reserved to indicate an error.

18732 **ERRORS**

18733 No errors are defined.

18734 **EXAMPLES**

18735 None.

18736 **APPLICATION USAGE**

18737 None.

18738 **RATIONALE**

18739 None.

18740 **FUTURE DIRECTIONS**

18741 None.

18742 **SEE ALSO**

18743 *ctime*( ), *ftime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/time.h>**

18744 **CHANGE HISTORY**

18745 First released in Issue 4, Version 2.

18746 **Issue 5**

18747 Moved from X/OPEN UNIX extension to BASE.

18748 **Issue 6**

18749 The DESCRIPTION is updated to refer to ‘‘seconds since the Epoch’’ rather than ‘‘seconds since
18750 00:00:00 UTC (Coordinated Universal Time), January 1 1970’’ for consistency with other *time*
18751 functions.

18752 The **restrict** keyword is added to the *gettimeofday*( ) prototype for alignment with the
18753 ISO/IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

getuid — get a real user ID

SYNOPSIS

```
#include <unistd.h>

uid_t getuid(void);
```

DESCRIPTION

The *getuid*( ) function shall return the real user ID of the calling process.

RETURN VALUE

The *getuid*( ) function shall always be successful and no return value is reserved to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

Setting the Effective User ID to the Real User ID

The following example sets the effective user ID and the real user ID of the current process to the real user ID of the caller.

```
#include <unistd.h>
#include <sys/types.h>
...
setreuid(getuid(), getuid());
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *geteuid*( ), *getgid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18793 **NAME**

18794 getutxent, getutxid, getutxline — get user accounting database entries

18795 **SYNOPSIS**

18796 XSI
```
#include <utmpx.h>

struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
```
18800

18801 **DESCRIPTION**

18802 Refer to *endutxent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18803 **NAME**

18804 getwc — get a wide character from a stream

18805 **SYNOPSIS**

18806 `#include <stdio.h>`

18807 `#include <wchar.h>`

18808 `wint_t getwc(FILE *stream);`

18809 **DESCRIPTION**

18810 CX The functionality described on this reference page is aligned with the ISO C standard. Any
18811 conflict between the requirements described here and the ISO C standard is unintentional. This
18812 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18813 The *getwc*( ) function shall be equivalent to *fgetwc*( ), except that if it is implemented as a macro it
18814 may evaluate *stream* more than once, so the argument should never be an expression with side
18815 effects.

18816 **RETURN VALUE**

18817 Refer to *fgetwc*( ).

18818 **ERRORS**

18819 Refer to *fgetwc*( ).

18820 **EXAMPLES**

18821 None.

18822 **APPLICATION USAGE**

18823 Since it may be implemented as a macro, *getwc*( ) may treat incorrectly a *stream* argument with
18824 side effects. In particular, *getwc*(*f*++) does not necessarily work as expected. Therefore, use of
18825 this function is not recommended; *fgetwc*( ) should be used instead.

18826 **RATIONALE**

18827 None.

18828 **FUTURE DIRECTIONS**

18829 None.

18830 **SEE ALSO**

18831 *fgetwc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<wchar.h>**

18832 **CHANGE HISTORY**

18833 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
18834 draft.

18835 **Issue 5**

18836 The Optional Header (OH) marking is removed from **<stdio.h>**.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18837 **NAME**

18838 getwchar — get a wide character from a stdin stream

18839 **SYNOPSIS**

18840
```
#include <wchar.h>
```

18841
```
wint_t getwchar(void);
```

18842 **DESCRIPTION**

18843 CX The functionality described on this reference page is aligned with the ISO C standard. Any
18844 conflict between the requirements described here and the ISO C standard is unintentional. This
18845 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18846 The *getwchar*( ) function shall be equivalent to *getwc*(*stdin*).

18847 **RETURN VALUE**

18848 Refer to *fgetwc*( ).

18849 **ERRORS**

18850 Refer to *fgetwc*( ).

18851 **EXAMPLES**

18852 None.

18853 **APPLICATION USAGE**

18854 If the **wint_t** value returned by *getwchar*( ) is stored into a variable of type **wchar_t** and then
18855 compared against the **wint_t** macro WEOF, the result may be incorrect. Only the **wint_t** type is
18856 guaranteed to be able to represent any wide character and WEOF.

18857 **RATIONALE**

18858 None.

18859 **FUTURE DIRECTIONS**

18860 None.

18861 **SEE ALSO**

18862 *fgetwc*( ), *getwc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

18863 **CHANGE HISTORY**

18864 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
18865 draft.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18866 **NAME**

18867 getwd — get the current working directory pathname (**LEGACY**)

18868 **SYNOPSIS**

18869 XSI
```
#include <unistd.h>
```

18870
```
char *getwd(char *path_name);
```

18871

18872 **DESCRIPTION**

18873 The *getwd*( ) function shall determine an absolute pathname of the current working directory of
18874 the calling process, and copy a string containing that pathname into the array pointed to by the
18875 *path_name* argument.

18876 If the length of the pathname of the current working directory is greater than ({PATH_MAX}+1)
18877 including the null byte, *getwd*( ) shall fail and return a null pointer.

18878 **RETURN VALUE**

18879 Upon successful completion, a pointer to the string containing the absolute pathname of the
18880 current working directory shall be returned. Otherwise, *getwd*( ) shall return a null pointer and
18881 the contents of the array pointed to by *path_name* are undefined.

18882 **ERRORS**

18883 No errors are defined.

18884 **EXAMPLES**

18885 None.

18886 **APPLICATION USAGE**

18887 For applications portability, the *getcwd*( ) function should be used to determine the current
18888 working directory instead of *getwd*( ).

18889 **RATIONALE**

18890 Since the user cannot specify the length of the buffer passed to *getwd*( ), use of this function is
18891 discouraged. The length of a pathname described in {PATH_MAX} is file system-dependent and
18892 may vary from one mount point to another, or might even be unlimited. It is possible to
18893 overflow this buffer in such a way as to cause applications to fail, or possible system security
18894 violations.

18895 It is recommended that the *getcwd*( ) function should be used to determine the current working
18896 directory.

18897 **FUTURE DIRECTIONS**

18898 This function may be withdrawn in a future version.

18899 **SEE ALSO**

18900 *getcwd*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

18901 **CHANGE HISTORY**

18902 First released in Issue 4, Version 2.

18903 **Issue 5**

18904 Moved from X/OPEN UNIX extension to BASE.

18905 **Issue 6**

18906 This function is marked LEGACY.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

glob, globfree — generate pathnames matching a pattern

SYNOPSIS

```
#include <glob.h>

int glob(const char *restrict pattern, int flags,
    int(*errfunc)(const char *epath, int eerrno),
    glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

DESCRIPTION

The *glob*( ) function is a pathname generator that shall implement the rules defined in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern Matching Notation, with optional support for rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13.3, Patterns Used for Filename Expansion.

The structure type **glob_t** is defined in **<glob.h>** and includes at least the following members:

Member Type	Member Name	Description
size_t	*gl_pathc*	Count of paths matched by *pattern*.
**char ** **	*gl_pathv*	Pointer to a list of matched pathnames.
size_t	*gl_offs*	Slots to reserve at the beginning of *gl_pathv*.

The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob*( ) function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, *glob*( ) requires search permission on every component of a path except the last, and read permission on each directory of any filename component of *pattern* that contains any of the following special characters: `'*'`, `'?'`, and `'['`.

The *glob*( ) function shall store the number of matched pathnames into *pglob->gl_pathc* and a pointer to a list of pointers to pathnames into *pglob->gl_pathv*. The pathnames shall be in sort order as defined by the current setting of the *LC_COLLATE* category; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.2, LC_COLLATE. The first pointer after the last pathname shall be a null pointer. If the pattern does not match any pathnames, the returned number of matched paths is set to 0, and the contents of *pglob->gl_pathv* are implementation-defined.

It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob*( ) function shall allocate other space as needed, including the memory pointed to by *gl_pathv*. The *globfree*( ) function shall free any space associated with *pglob* from a previous call to *glob*( ).

The *flags* argument is used to control the behavior of *glob*( ). The value of *flags* is a bitwise-inclusive OR of zero or more of the following constants, which are defined in **<glob.h>**:

GLOB_APPEND — Append pathnames generated to the ones from a previous call to *glob*( ).

GLOB_DOOFFS — Make use of *pglob->gl_offs*. If this flag is set, *pglob->gl_offs* is used to specify how many null pointers to add to the beginning of *pglob->gl_pathv*. In other words, *pglob->gl_pathv* shall point to *pglob->gl_offs* null pointers, followed by *pglob->gl_pathc* pathname pointers, followed by a null pointer.

GLOB_ERR — Cause *glob*( ) to return when it encounters a directory that it cannot open or read. Ordinarily, *glob*( ) continues to find matches.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

GLOB_MARK — Each pathname that is a directory that matches *pattern* shall have a slash appended.

GLOB_NOCHECK — Supports rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13.3, Patterns Used for Filename Expansion. If *pattern* does not match any pathname, then *glob*( ) shall return a list consisting of only *pattern*, and the number of matched pathnames is 1.

GLOB_NOESCAPE — Disable backslash escaping.

GLOB_NOSORT — Ordinarily, *glob*( ) sorts the matching pathnames according to the current setting of the *LC_COLLATE* category; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.2, LC_COLLATE. When this flag is used, the order of pathnames returned is unspecified.

The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a previous call to *glob*( ). The following rules apply to applications when two or more calls to *glob*( ) are made with the same value of *pglob* and without intervening calls to *globfree*( ):

1. The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.
2. All the calls shall set GLOB_DOOFFS, or all shall not set it.
3. After the second call, *pglob->gl_pathv* points to a list containing the following:
   a. Zero or more null pointers, as specified by GLOB_DOOFFS and *pglob->gl_offs*.
   b. Pointers to the pathnames that were in the *pglob->gl_pathv* list before the call, in the same order as before.
   c. Pointers to the new pathnames generated by the second call, in the specified order.
4. The count returned in *pglob->gl_pathc* shall be the total number of pathnames from the two calls.
5. The application can change any of the fields after a call to *glob*( ). If it does, the application shall reset them to the original value before a subsequent call, using the same *pglob* value, to *globfree*( ) or *glob*( ) with the GLOB_APPEND flag.

If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not a null pointer, *glob*( ) calls (**errfunc*( )) with two arguments:

1. The *epath* argument is a pointer to the path that failed.
2. The *eerrno* argument is the value of *errno* from the failure, as set by *opendir*( ), *readdir*( ), or *stat*( ). (Other values may be used to report other errors not explicitly documented for those functions.)

If (**errfunc*( )) is called and returns non-zero, or if the GLOB_ERR flag is set in *flags*, *glob*( ) shall stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect the paths already scanned. If GLOB_ERR is not set and either *errfunc* is a null pointer or (**errfunc*( )) returns 0, the error shall be ignored.

The *glob*( ) function shall not fail because of large files.

RETURN VALUE

Upon successful completion, *glob*( ) shall return 0. The argument *pglob->gl_pathc* shall return the number of matched pathnames and the argument *pglob->gl_pathv* shall contain a pointer to a null-terminated list of matched and sorted pathnames. However, if *pglob->gl_pathc* is 0, the content of *pglob->gl_pathv* is undefined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *globfree*( ) function shall not return a value.

If *glob*( ) terminates due to an error, it shall return one of the non-zero constants defined in **<glob.h>**. The arguments *pglob->gl_pathc* and *pglob->gl_pathv* are still set as defined above.

ERRORS

The *glob*( ) function shall fail and return the corresponding value if:

GLOB_ABORTED — The scan was stopped because GLOB_ERR was set or (**errfunc*( )) returned non-zero.

GLOB_NOMATCH — The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in flags.

GLOB_NOSPACE — An attempt to allocate memory failed.

EXAMPLES

One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with *execv*( ), *execve*( ), or *execvp*( ). Suppose, for example, that an application wants to do the equivalent of:

```
ls -l *.c
```

but for some reason:

```
system("ls -l *.c")
```

is not acceptable. The application could obtain approximately the same result using the sequence:

```
globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```
ls -l *.c *.h
```

could be approximately simulated using GLOB_APPEND as follows:

```
globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
...
```

APPLICATION USAGE

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch*( ).

Note that *gl_pathc* and *gl_pathv* have meaning even if *glob*( ) fails. This allows *glob*( ) to report partial results in the event of an error. However, if *gl_pathc* is 0, *gl_pathv* is unspecified even if *glob*( ) did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

might use this for option-arguments, for example.

The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname expansion when multiple expansions are done on a command line.

Applications that need tilde and parameter expansion should use *wordexp*( ).

RATIONALE

It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
       * sizeof(char *));
(void) memcpy(new+n, pglob->gl_pathv,
       pglob->gl_pathc * sizeof(char *));
(void) memset(new, 0, n * sizeof(char *));
free(pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc*( ). This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc*( ) in a large chunk and then carved up within *glob*( ), or it might have been created using a different memory allocator. It is not the intent of the standard developers to specify or imply how the memory used by *glob*( ) is managed.

The GLOB_APPEND flag would be used when an application wants to expand several different patterns into a single list.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fnmatch*( ), *opendir*( ), *readdir*( ), *stat*( ), *wordexp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<glob.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *glob*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

gmtime, gmtime_r — convert a time value to a broken-down UTC time

SYNOPSIS

```
#include <time.h>

struct tm *gmtime(const time_t *timer);
TSF struct tm *gmtime_r(const time_t *restrict timer,
        struct tm *restrict result);
```

DESCRIPTION

CX For *gmtime*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *gmtime*( ) function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time, expressed as Coordinated Universal Time (UTC).

CX The relationship between a time in seconds since the Epoch used as an argument to *gmtime*( ) and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch), where the names in the structure and in the expression correspond.

TSF The same relationship shall apply for *gmtime_r*( ).

CX The *gmtime*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

The *asctime*( ), *ctime*( ), *gmtime*( ), and *localtime*( ) functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

TSF The *gmtime_r*( ) function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down time is stored in the structure referred to by *result*. The *gmtime_r*( ) function shall also return the address of the same structure.

RETURN VALUE

Upon successful completion, the *gmtime*( ) function shall return a pointer to a **struct tm**. If an
CX error is detected, *gmtime*( ) shall return a null pointer and set *errno* to indicate the error.

TSF Upon successful completion, *gmtime_r*( ) shall return the address of the structure pointed to by the argument *result*. If an error is detected, *gmtime_r*( ) shall return a null pointer.

ERRORS

The *gmtime*( ) function shall fail if:

CX [EOVERFLOW] The result cannot be represented.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The *gmtime_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *gmtime*( ) function need not be reentrant is added to the DESCRIPTION.

The *gmtime_r*( ) function is included for alignment with the POSIX Threads Extension.

Issue 6

The *gmtime_r*( ) function is marked as part of the Thread-Safe Functions option.

Extensions beyond the ISO C standard are marked.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *gmtime_r*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [EOVERFLOW] error.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19136 **NAME**

19137 grantpt — grant access to the slave pseudo-terminal device

19138 **SYNOPSIS**

19139 XSI
```
#include <stdlib.h>

int grantpt(int fildes);
```
19140

19141

19142 **DESCRIPTION**

19143 The *grantpt*( ) function shall change the mode and ownership of the slave pseudo-terminal
19144 device associated with its master pseudo-terminal counterpart. The *fildes* argument is a file
19145 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to
19146 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The
19147 permission mode of the slave pseudo-terminal shall be set to readable and writable by the
19148 owner, and writable by the group.

19149 The behavior of the *grantpt*( ) function is unspecified if the application has installed a signal
19150 handler to catch SIGCHLD signals.

19151 **RETURN VALUE**

19152 Upon successful completion, *grantpt*( ) shall return 0; otherwise, it shall return –1 and set *errno* to
19153 indicate the error.

19154 **ERRORS**

19155 The *grantpt*( ) function may fail if:

19156 [EBADF] The *fildes* argument is not a valid open file descriptor.

19157 [EINVAL] The *fildes* argument is not associated with a master pseudo-terminal device.

19158 [EACCES] The corresponding slave pseudo-terminal device could not be accessed.

19159 **EXAMPLES**

19160 None.

19161 **APPLICATION USAGE**

19162 None.

19163 **RATIONALE**

19164 None.

19165 **FUTURE DIRECTIONS**

19166 None.

19167 **SEE ALSO**

19168 *open*( ), *ptsname*( ), *unlockpt*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

19169 **CHANGE HISTORY**

19170 First released in Issue 4, Version 2.

19171 **Issue 5**

19172 Moved from X/OPEN UNIX extension to BASE.

19173 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19174 **NAME**

19175 h_errno — error return value for network database operations

19176 **SYNOPSIS**

19177 OB
```
#include <netdb.h>
```
19178

19179 **DESCRIPTION**

19180 This method of returning errors is used only in connection with obsolescent functions.

19181 The **<netdb.h>** header provides a declaration of *h_errno* as a modifiable lvalue of type **int**.

19182 It is unspecified whether *h_errno* is a macro or an identifier declared with external linkage. If a
19183 macro definition is suppressed in order to access an actual object, or a program defines an
19184 identifier with the name *h_errno*, the behavior is undefined.

19185 **RETURN VALUE**

19186 None.

19187 **ERRORS**

19188 No errors are defined.

19189 **EXAMPLES**

19190 None.

19191 **APPLICATION USAGE**

19192 Applications should obtain the definition of *h_errno* by the inclusion of the **<netdb.h>** header.

19193 **RATIONALE**

19194 None.

19195 **FUTURE DIRECTIONS**

19196 *h_errno* may be withdrawn in a future version.

19197 **SEE ALSO**

19198 *endhostent*( ), *errno*, the Base Definitions volume of IEEE Std 1003.1-2001, **<netdb.h>**

19199 **CHANGE HISTORY**

19200 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19201 **NAME**

19202 hcreate, hdestroy, hsearch — manage hash search table

19203 **SYNOPSIS**

19204 XSI
```
#include <search.h>

int hcreate(size_t nel);
void hdestroy(void);
ENTRY *hsearch(ENTRY item, ACTION action);
```
19208

19209 **DESCRIPTION**

19210 The *hcreate*( ), *hdestroy*( ), and *hsearch*( ) functions shall manage hash search tables.

19211 The *hcreate*( ) function shall allocate sufficient space for the table, and the application shall
19212 ensure it is called before *hsearch*( ) is used. The *nel* argument is an estimate of the maximum
19213 number of entries that the table shall contain. This number may be adjusted upward by the
19214 algorithm in order to obtain certain mathematically favorable circumstances.

19215 The *hdestroy*( ) function shall dispose of the search table, and may be followed by another call to
19216 *hcreate*( ). After the call to *hdestroy*( ), the data can no longer be considered accessible.

19217 The *hsearch*( ) function is a hash-table search routine. It shall return a pointer into a hash table
19218 indicating the location at which an entry can be found. The *item* argument is a structure of type
19219 **ENTRY** (defined in the **<search.h>** header) containing two pointers: *item.key* points to the
19220 comparison key (a **char ***), and *item.data* (a **void ***) points to any other data to be associated with
19221 that key. The comparison function used by *hsearch*( ) is *strcmp*( ). The *action* argument is a
19222 member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be
19223 found in the table. ENTER indicates that the item should be inserted in the table at an
19224 appropriate point. FIND indicates that no entry should be made. Unsuccessful resolution is
19225 indicated by the return of a null pointer.

19226 These functions need not be reentrant. A function that is not required to be reentrant is not
19227 required to be thread-safe.

19228 **RETURN VALUE**

19229 The *hcreate*( ) function shall return 0 if it cannot allocate sufficient space for the table; otherwise,
19230 it shall return non-zero.

19231 The *hdestroy*( ) function shall not return a value.

19232 The *hsearch*( ) function shall return a null pointer if either the action is FIND and the item could
19233 not be found or the action is ENTER and the table is full.

19234 **ERRORS**

19235 The *hcreate*( ) and *hsearch*( ) functions may fail if:

19236 [ENOMEM] Insufficient storage space is available.

19237 **EXAMPLES**

19238 The following example reads in strings followed by two numbers and stores them in a hash
19239 table, discarding duplicates. It then reads in strings and finds the matching entry in the hash
19240 table and prints it out.

```
19241 #include <stdio.h>
19242 #include <search.h>
19243 #include <string.h>

19244 struct info {        /* This is the info stored in the table */
19245     int age, room;   /* other than the key. */
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
};

#define NUM_EMPL    5000    /* # of elements in search table. */

int main(void)
{
    char string_space[NUM_EMPL*20];   /* Space to store strings. */
    struct info info_space[NUM_EMPL]; /* Space to store employee info. */
    char *str_ptr = string_space;     /* Next space in string_space. */
    struct info *info_ptr = info_space;
                                      /* Next space in info_space. */
    ENTRY item;
    ENTRY *found_item; /* Name to look for in table. */
    char name_to_find[30];

    int i = 0;

    /* Create table; no error checking is performed. */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
           &info_ptr->room) != EOF && i++ < NUM_EMPL) {

        /* Put information in structure, and structure in item. */
        item.key = str_ptr;
        item.data = info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;

        /* Put item into table. */
        (void) hsearch(item, ENTER);
    }

    /* Access table. */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if ((found_item = hsearch(item, FIND)) != NULL) {

            /* If item is in the table. */
            (void)printf("found %s, age = %d, room = %d\n",
                found_item->key,
                ((struct info *)found_item->data)->age,
                ((struct info *)found_item->data)->room);
        } else
            (void)printf("no such employee %s\n", name_to_find);
    }
    return 0;
}
```

APPLICATION USAGE

The *hcreate*( ) and *hsearch*( ) functions may use *malloc*( ) to allocate space.

RATIONALE

None.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19289 **FUTURE DIRECTIONS**

19290 None.

19291 **SEE ALSO**

19292 *bsearch*( ), *lsearch*( ), *malloc*( ), *strcmp*( ), *tsearch*( ), the Base Definitions volume of
19293 IEEE Std 1003.1-2001, **<search.h>**

19294 **CHANGE HISTORY**

19295 First released in Issue 1. Derived from Issue 1 of the SVID.

19296 **Issue 6**

19297 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

19298 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`--`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

19299 **NAME**

19300 htonl, htons, ntohl, ntohs — convert values between host and network byte order

19301 **SYNOPSIS**

19302
```
#include <arpa/inet.h>
```

19303
```
uint32_t htonl(uint32_t hostlong);
```
19304
```
uint16_t htons(uint16_t hostshort);
```
19305
```
uint32_t ntohl(uint32_t netlong);
```
19306
```
uint16_t ntohs(uint16_t netshort);
```

19307 **DESCRIPTION**

19308 These functions shall convert 16-bit and 32-bit quantities between network byte order and host
19309 byte order.

19310 On some implementations, these functions are defined as macros.

19311 The **uint32_t** and **uint16_t** types are defined in **<inttypes.h>**.

19312 **RETURN VALUE**

19313 The *htonl*( ) and *htons*( ) functions shall return the argument value converted from host to
19314 network byte order.

19315 The *ntohl*( ) and *ntohs*( ) functions shall return the argument value converted from network to
19316 host byte order.

19317 **ERRORS**

19318 No errors are defined.

19319 **EXAMPLES**

19320 None.

19321 **APPLICATION USAGE**

19322 These functions are most often used in conjunction with IPv4 addresses and ports as returned by
19323 *gethostent*( ) and *getservent*( ).

19324 **RATIONALE**

19325 None.

19326 **FUTURE DIRECTIONS**

19327 None.

19328 **SEE ALSO**

19329 *endhostent*( ), *endservent*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<inttypes.h>**,
19330 **<arpa/inet.h>**

19331 **CHANGE HISTORY**

19332 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19333 **NAME**

19334 hypot, hypotf, hypotl — Euclidean distance function

19335 **SYNOPSIS**

19336
```
#include <math.h>

double hypot(double x, double y);
float hypotf(float x, float y);
long double hypotl(long double x, long double y);
```

19340 **DESCRIPTION**

19341 CX The functionality described on this reference page is aligned with the ISO C standard. Any
19342 conflict between the requirements described here and the ISO C standard is unintentional. This
19343 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19344 These functions shall compute the value of the square root of x^2+y^2 without undue overflow or
19345 underflow.

19346 An application wishing to check for error situations should set *errno* to zero and call
19347 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
19348 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
19349 zero, an error has occurred.

19350 **RETURN VALUE**

19351 Upon successful completion, these functions shall return the length of the hypotenuse of a
19352 right-angled triangle with sides of length *x* and *y*.

19353 If the correct value would cause overflow, a range error shall occur and *hypot*( ), *hypotf*( ), and
19354 *hypotl*( ) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
19355 respectively.

19356 MX If *x* or *y* is ±Inf, +Inf shall be returned (even if one of *x* or *y* is NaN).

19357 If *x* or *y* is NaN, and the other is not ±Inf, a NaN shall be returned.

19358 If both arguments are subnormal and the correct result is subnormal, a range error may occur
19359 and the correct result is returned.

19360 **ERRORS**

19361 These functions shall fail if:

19362 Range Error The result overflows.

19363 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
19364 then *errno* shall be set to [ERANGE]. If the integer expression
19365 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
19366 floating-point exception shall be raised.

19367 These functions may fail if:

19368 MX Range Error The result underflows.

19369 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
19370 then *errno* shall be set to [ERANGE]. If the integer expression
19371 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
19372 floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

hypot(*x*,*y*), *hypot*(*y*,*x*), and *hypot*(*x*, –*y*) are equivalent.

hypot(*x*, ±0) is equivalent to *fabs*(*x*).

Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also subnormal.

These functions take precautions against overflow during intermediate steps of the computation.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *sqrt*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *hypot*( ) function is no longer marked as an extension.

The *hypotf*( ) and *hypotl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19404 **NAME**

19405 iconv — codeset conversion function

19406 **SYNOPSIS**

19407 XSI
```
#include <iconv.h>

size_t iconv(iconv_t cd, char **restrict inbuf,
    size_t *restrict inbytesleft, char **restrict outbuf,
    size_t *restrict outbytesleft);
```
19411

19412 **DESCRIPTION**

19413 The *iconv*( ) function shall convert the sequence of characters from one codeset, in the array
19414 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array
19415 specified by *outbuf*. The codesets are those specified in the *iconv_open*( ) call that returned the
19416 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first
19417 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer
19418 to be converted. The *outbuf* argument points to a variable that points to the first available byte in
19419 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the
19420 buffer.

19421 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by
19422 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv*( ) is
19423 called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*
19424 points to a positive value, *iconv*( ) shall place, into the output buffer, the byte sequence to change
19425 the output buffer to its initial shift state. If the output buffer is not large enough to hold the
19426 entire reset sequence, *iconv*( ) shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as
19427 other than a null pointer or a pointer to a null pointer cause the conversion to take place from
19428 the current state of the conversion descriptor.

19429 If a sequence of input bytes does not form a valid character in the specified codeset, conversion
19430 shall stop after the previous successfully converted character. If the input buffer ends with an
19431 incomplete character or shift sequence, conversion shall stop after the previous successfully
19432 converted bytes. If the output buffer is not large enough to hold the entire converted input,
19433 conversion shall stop just prior to the input bytes that would cause the output buffer to
19434 overflow. The variable pointed to by *inbuf* shall be updated to point to the byte following the last
19435 byte successfully used in the conversion. The value pointed to by *inbytesleft* shall be
19436 decremented to reflect the number of bytes still not converted in the input buffer. The variable
19437 pointed to by *outbuf* shall be updated to point to the byte following the last byte of converted
19438 output data. The value pointed to by *outbytesleft* shall be decremented to reflect the number of
19439 bytes still available in the output buffer. For state-dependent encodings, the conversion
19440 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully
19441 converted byte sequence.

19442 If *iconv*( ) encounters a character in the input buffer that is valid, but for which an identical
19443 character does not exist in the target codeset, *iconv*( ) shall perform an implementation-defined
19444 conversion on this character.

19445 **RETURN VALUE**

19446 The *iconv*( ) function shall update the variables pointed to by the arguments to reflect the extent
19447 of the conversion and return the number of non-identical conversions performed. If the entire
19448 string in the input buffer is converted, the value pointed to by *inbytesleft* shall be 0. If the input
19449 conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft*
19450 shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs, *iconv*( ) shall
19451 return (**size_t**)–1 and set *errno* to indicate the error.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19452 **ERRORS**

19453 The *iconv*( ) function shall fail if:

19454 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the
19455 input codeset.

19456 [E2BIG] Input conversion stopped due to lack of space in the output buffer.

19457 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at
19458 the end of the input buffer.

19459 The *iconv*( ) function may fail if:

19460 [EBADF] The *cd* argument is not a valid open conversion descriptor.

19461 **EXAMPLES**

19462 None.

19463 **APPLICATION USAGE**

19464 The *inbuf* argument indirectly points to the memory area which contains the conversion input
19465 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of
19466 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to
19467 containing data that is directly representable in the ISO C standard language **char** data type. The
19468 type of *inbuf* and *outbuf*, **char ****, does not imply that the objects pointed to are interpreted as
19469 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that
19470 represents a character in a given character set encoding scheme is done internally within the
19471 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can
19472 contain all zero octets that are not interpreted as string terminators but as coded character data
19473 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and
19474 so on) read or stored in the objects is not specified, but may be inferred for both the input and
19475 output data by the converters determined by the *fromcode* and *tocode* arguments of *iconv_open*( ).

19476 Regardless of the data type inferred by the converter, the size of the remaining space in both
19477 input and output objects (the *intbytesleft* and *outbytesleft* arguments) is always measured in bytes.

19478 For implementations that support the conversion of state-dependent encodings, the conversion
19479 descriptor must be able to accurately reflect the shift-state in effect at the end of the last
19480 successful conversion. It is not required that the conversion descriptor itself be updated, which
19481 would require it to be a pointer type. Thus, implementations are free to implement the
19482 descriptor as a handle (other than a pointer type) by which the conversion information can be
19483 accessed and updated.

19484 **RATIONALE**

19485 None.

19486 **FUTURE DIRECTIONS**

19487 None.

19488 **SEE ALSO**

19489 *iconv_open*( ), *iconv_close*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<iconv.h>**

19490 **CHANGE HISTORY**

19491 First released in Issue 4. Derived from the HP-UX Manual.

19492 **Issue 6**

19493 The SYNOPSIS has been corrected to align with the **<iconv.h>** reference page.

19494 The **restrict** keyword is added to the *iconv*( ) prototype for alignment with the
19495 ISO/IEC 9899:1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19496 **NAME**

19497 iconv_close — codeset conversion deallocation function

19498 **SYNOPSIS**

19499 XSI
```
#include <iconv.h>
```

19500
```
int iconv_close(iconv_t cd);
```

19501

19502 **DESCRIPTION**

19503 The *iconv_close*( ) function shall deallocate the conversion descriptor *cd* and all other associated
19504 resources allocated by *iconv_open*( ).

19505 If a file descriptor is used to implement the type **iconv_t**, that file descriptor shall be closed.

19506 **RETURN VALUE**

19507 Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to
19508 indicate the error.

19509 **ERRORS**

19510 The *iconv_close*( ) function may fail if:

19511 [EBADF] The conversion descriptor is invalid.

19512 **EXAMPLES**

19513 None.

19514 **APPLICATION USAGE**

19515 None.

19516 **RATIONALE**

19517 None.

19518 **FUTURE DIRECTIONS**

19519 None.

19520 **SEE ALSO**

19521 *iconv*( ), *iconv_open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<iconv.h>**

19522 **CHANGE HISTORY**

19523 First released in Issue 4. Derived from the HP-UX Manual.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

iconv_open — codeset conversion allocation function

SYNOPSIS

XSI

```
#include <iconv.h>

iconv_t iconv_open(const char *tocode, const char *fromcode);
```

DESCRIPTION

The *iconv_open*( ) function shall return a conversion descriptor that describes a conversion from the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with *iconv*( ).

Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.

A conversion descriptor shall remain valid until it is closed by *iconv_close*( ) or an implicit close.

If a file descriptor is used to implement conversion descriptors, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

RETURN VALUE

Upon successful completion, *iconv_open*( ) shall return a conversion descriptor for use on subsequent calls to *iconv*( ). Otherwise, *iconv_open*( ) shall return (**iconv_t**)–1 and set *errno* to indicate the error.

ERRORS

The *iconv_open*( ) function may fail if:

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENFILE] Too many files are currently open in the system.

[ENOMEM] Insufficient storage space is available.

[EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the implementation.

EXAMPLES

None.

APPLICATION USAGE

Some implementations of *iconv_open*( ) use *malloc*( ) to allocate space for internal buffer areas. The *iconv_open*( ) function may fail if there is insufficient storage space to accommodate these buffers.

Conforming applications must assume that conversion descriptors are not valid after a call to one of the *exec* functions.

Application developers should consult the system documentation to determine the supported codesets and their naming schemes.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19565 **SEE ALSO**

19566 *iconv*( ), *iconv_close*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<iconv.h>**

19567 **CHANGE HISTORY**

19568 First released in Issue 4. Derived from the HP-UX Manual.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19569 **NAME**

19570 if_freenameindex — free memory allocated by if_nameindex

19571 **SYNOPSIS**

19572
```
#include <net/if.h>
```

19573
```
void if_freenameindex(struct if_nameindex *ptr);
```

19574 **DESCRIPTION**

19575 The *if_freenameindex*( ) function shall free the memory allocated by *if_nameindex*( ). The *ptr*
19576 argument shall be a pointer that was returned by *if_nameindex*( ). After *if_freenameindex*( ) has
19577 been called, the application shall not use the array of which *ptr* is the address.

19578 **RETURN VALUE**

19579 None.

19580 **ERRORS**

19581 No errors are defined.

19582 **EXAMPLES**

19583 None.

19584 **APPLICATION USAGE**

19585 None.

19586 **RATIONALE**

19587 None.

19588 **FUTURE DIRECTIONS**

19589 None.

19590 **SEE ALSO**

19591 *getsockopt*( ), *if_indextoname*( ), *if_nameindex*( ), *if_nametoindex*( ), *setsockopt*( ), the Base Definitions
19592 volume of IEEE Std 1003.1-2001, **<net/if.h>**

19593 **CHANGE HISTORY**

19594 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19595 **NAME**

19596 if_indextoname — map a network interface index to its corresponding name

19597 **SYNOPSIS**

19598 `#include <net/if.h>`

19599 `char *if_indextoname(unsigned ifindex, char *ifname);`

19600 **DESCRIPTION**

19601 The *if_indextoname*( ) function shall map an interface index to its corresponding name.

19602 When this function is called, *ifname* shall point to a buffer of at least {IF_NAMESIZE} bytes. The |
19603 function shall place in this buffer the name of the interface with index *ifindex*.

19604 **RETURN VALUE**

19605 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which
19606 points to a buffer now containing the interface name. Otherwise, the function shall return a
19607 NULL pointer and set *errno* to indicate the error.

19608 **ERRORS**

19609 The *if_indextoname*( ) function shall fail if:

19610 [ENXIO] The interface does not exist.

19611 **EXAMPLES**

19612 None.

19613 **APPLICATION USAGE**

19614 None.

19615 **RATIONALE**

19616 None.

19617 **FUTURE DIRECTIONS**

19618 None.

19619 **SEE ALSO**

19620 *getsockopt*( ), *if_freenameindex*( ), *if_nameindex*( ), *if_nametoindex*( ), *setsockopt*( ), the Base
19621 Definitions volume of IEEE Std 1003.1-2001, **<net/if.h>**

19622 **CHANGE HISTORY**

19623 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19624 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to |
19625 {IF_NAMESIZ} in the DESCRIPTION. |

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19626 **NAME**

19627 if_nameindex — return all network interface names and indexes

19628 **SYNOPSIS**

19629
```
#include <net/if.h>
```

19630
```
struct if_nameindex *if_nameindex(void);
```

19631 **DESCRIPTION**

19632 The *if_nameindex*( ) function shall return an array of *if_nameindex* structures, one structure per
19633 interface. The end of the array is indicated by a structure with an *if_index* field of zero and an
19634 *if_name* field of NULL.

19635 Applications should call *if_freenameindex*( ) to release the memory that may be dynamically
19636 allocated by this function, after they have finished using it.

19637 **RETURN VALUE**

19638 An array of structures identifying local interfaces. A NULL pointer is returned upon an error,
19639 with *errno* set to indicate the error.

19640 **ERRORS**

19641 The *if_nameindex*( ) function may fail if:

19642 [ENOBUFS] Insufficient resources are available to complete the function.

19643 **EXAMPLES**

19644 None.

19645 **APPLICATION USAGE**

19646 None.

19647 **RATIONALE**

19648 None.

19649 **FUTURE DIRECTIONS**

19650 None.

19651 **SEE ALSO**

19652 *getsockopt*( ), *if_freenameindex*( ), *if_indextoname*( ), *if_nametoindex*( ), *setsockopt*( ), the Base
19653 Definitions volume of IEEE Std 1003.1-2001, **<net/if.h>**

19654 **CHANGE HISTORY**

19655 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19656 **NAME**

19657 if_nametoindex — map a network interface name to its corresponding index

19658 **SYNOPSIS**

19659 `#include <net/if.h>`

19660 `unsigned if_nametoindex(const char *`*`ifname`*`);`

19661 **DESCRIPTION**

19662 The *if_nametoindex*( ) function shall return the interface index corresponding to name *ifname*.

19663 **RETURN VALUE**

19664 The corresponding index if *ifname* is the name of an interface; otherwise, zero.

19665 **ERRORS**

19666 No errors are defined.

19667 **EXAMPLES**

19668 None.

19669 **APPLICATION USAGE**

19670 None.

19671 **RATIONALE**

19672 None.

19673 **FUTURE DIRECTIONS**

19674 None.

19675 **SEE ALSO**

19676 *getsockopt*( ), *if_freenameindex*( ), *if_indextoname*( ), *if_nameindex*( ), *setsockopt*( ), the Base
19677 Definitions volume of IEEE Std 1003.1-2001, **<net/if.h>**

19678 **CHANGE HISTORY**

19679 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ilogb, ilogbf, ilogbl — return an unbiased exponent

SYNOPSIS

```
#include <math.h>

int ilogb(double x);
int ilogbf(float x);
int ilogbl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall return the exponent part of their argument *x*. Formally, the return value is the integral part of $\log_r |x|$ as a signed integral value, for non-zero *x*, where *r* is the radix of the machine's floating-point arithmetic, which is the value of FLT_RADIX defined in **<float.h>**.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the exponent part of *x* as a signed integer value. They are equivalent to calling the corresponding *logb*( ) function and casting the returned value to type **int**.

XSI If *x* is 0, a domain error shall occur, and the value FP_ILOGB0 shall be returned.

XSI If *x* is ±Inf, a domain error shall occur, and the value {INT_MAX} shall be returned.

XSI If *x* is a NaN, a domain error shall occur, and the value FP_ILOGBNAN shall be returned.

XSI If the correct value is greater than {INT_MAX}, {INT_MAX} shall be returned and a domain error shall occur.

If the correct value is less than {INT_MIN}, {INT_MIN} shall be returned and a domain error shall occur.

ERRORS

These functions shall fail if:

XSI

Domain Error	The *x* argument is zero, NaN, or ±Inf, or the correct value is not representable as an integer.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19717 **EXAMPLES**

19718 None.

19719 **APPLICATION USAGE**

19720 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
19721 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19722 **RATIONALE**

19723 The errors come from taking the expected floating-point value and converting it to **int**, which is
19724 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not
19725 representable in a type **int**), so should be a domain error.

19726 There are no known implementations that overflow. For overflow to happen, {INT_MAX} must
19727 be less than LDBL_MAX_EXP**log2*(FLT_RADIX) or {INT_MIN} must be greater than
19728 LDBL_MIN_EXP**log2*(FLT_RADIX) if subnormals are not supported, or {INT_MIN} must be
19729 greater than (LDBL_MIN_EXP-LDBL_MANT_DIG)**log2*(FLT_RADIX) if subnormals are
19730 supported.

19731 **FUTURE DIRECTIONS**

19732 None.

19733 **SEE ALSO**

19734 *feclearexcept*( ), *fetestexcept*( ), *logb*( ), *scalb*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
19735 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<float.h>**, **<math.h>**

19736 **CHANGE HISTORY**

19737 First released in Issue 4, Version 2.

19738 **Issue 5**

19739 Moved from X/OPEN UNIX extension to BASE.

19740 **Issue 6**

19741 The *ilogb*( ) function is no longer marked as an extension.

19742 The *ilogbf*( ) and *ilogbl*( ) functions are added for alignment with the ISO/IEC 9899:1999
19743 standard.

19744 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

19745 XSI extensions are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19746 **NAME**

19747 imaxabs — return absolute value

19748 **SYNOPSIS**

19749
```
#include <inttypes.h>
```

19750
```
intmax_t imaxabs(intmax_t j);
```

19751 **DESCRIPTION**

19752 CX The functionality described on this reference page is aligned with the ISO C standard. Any
19753 conflict between the requirements described here and the ISO C standard is unintentional. This
19754 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19755 The *imaxabs*( ) function shall compute the absolute value of an integer *j*. If the result cannot be
19756 represented, the behavior is undefined.

19757 **RETURN VALUE**

19758 The *imaxabs*( ) function shall return the absolute value.

19759 **ERRORS**

19760 No errors are defined.

19761 **EXAMPLES**

19762 None.

19763 **APPLICATION USAGE**

19764 The absolute value of the most negative number cannot be represented in two's complement.

19765 **RATIONALE**

19766 None.

19767 **FUTURE DIRECTIONS**

19768 None.

19769 **SEE ALSO**

19770 *imaxdiv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<inttypes.h>**

19771 **CHANGE HISTORY**

19772 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

imaxdiv — return quotient and remainder

SYNOPSIS

```
#include <inttypes.h>

imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *imaxdiv*( ) function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.

RETURN VALUE

The *imaxdiv*( ) function shall return a structure of type **imaxdiv_t**, comprising both the quotient and the remainder. The structure shall contain (in either order) the members *quot* (the quotient) and *rem* (the remainder), each of which has type **intmax_t**.

If either part of the result cannot be represented, the behavior is undefined.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

imaxabs( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<inttypes.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

19802 **NAME**

19803 index — character string operations (**LEGACY**)

19804 **SYNOPSIS**

19805 XSI
```
#include <strings.h>
```

19806
```
char *index(const char *s, int c);
```

19807

19808 **DESCRIPTION**

19809 The *index*( ) function shall be equivalent to *strchr*( ).

19810 **RETURN VALUE**

19811 See *strchr*( ).

19812 **ERRORS**

19813 See *strchr*( ).

19814 **EXAMPLES**

19815 None.

19816 **APPLICATION USAGE**

19817 The *strchr*( ) function is preferred over this function.

19818 For maximum portability, it is recommended to replace the function call to *index*( ) as follows:

19819
```
#define index(a,b) strchr((a),(b))
```

19820 **RATIONALE**

19821 None.

19822 **FUTURE DIRECTIONS**

19823 This function may be withdrawn in a future version.

19824 **SEE ALSO**

19825 *strchr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

19826 **CHANGE HISTORY**

19827 First released in Issue 4, Version 2.

19828 **Issue 5**

19829 Moved from X/OPEN UNIX extension to BASE.

19830 **Issue 6**

19831 This function is marked LEGACY.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19832 **NAME**

19833 inet_addr, inet_ntoa — IPv4 address manipulation

19834 **SYNOPSIS**

19835 `#include <arpa/inet.h>`

19836 `in_addr_t inet_addr(const char *`*`cp`*`);`
19837 `char *inet_ntoa(struct in_addr` *`in`*`);`

19838 **DESCRIPTION**

19839 The *inet_addr*( ) function shall convert the string pointed to by *cp*, in the standard IPv4 dotted
19840 decimal notation, to an integer value suitable for use as an Internet address.

19841 The *inet_ntoa*( ) function shall convert the Internet host address specified by *in* to a string in the
19842 Internet standard dot notation.

19843 The *inet_ntoa*( ) function need not be reentrant. A function that is not required to be reentrant is
19844 not required to be thread-safe.

19845 All Internet addresses shall be returned in network order (bytes ordered from left to right).

19846 Values specified using IPv4 dotted decimal notation take one of the following forms:

19847 `a.b.c.d` When four parts are specified, each shall be interpreted as a byte of data and
19848 assigned, from left to right, to the four bytes of an Internet address.

19849 `a.b.c` When a three-part address is specified, the last part shall be interpreted as a 16-bit
19850 quantity and placed in the rightmost two bytes of the network address. This makes
19851 the three-part address format convenient for specifying Class B network addresses
19852 as `"128.net.host"`.

19853 `a.b` When a two-part address is supplied, the last part shall be interpreted as a 24-bit
19854 quantity and placed in the rightmost three bytes of the network address. This
19855 makes the two-part address format convenient for specifying Class A network
19856 addresses as `"net.host"`.

19857 `a` When only one part is given, the value shall be stored directly in the network
19858 address without any byte rearrangement.

19859 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or
19860 hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal;
19861 otherwise, a leading `'0'` implies octal; otherwise, the number is interpreted as decimal).

19862 **RETURN VALUE**

19863 Upon successful completion, *inet_addr*( ) shall return the Internet address. Otherwise, it shall
19864 return (**in_addr_t**)(–1).

19865 The *inet_ntoa*( ) function shall return a pointer to the network address in Internet standard dot
19866 notation.

19867 **ERRORS**

19868 No errors are defined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19869 **EXAMPLES**

19870 None.

19871 **APPLICATION USAGE**

19872 The return value of *inet_ntoa*( ) may point to static data that may be overwritten by subsequent
19873 calls to *inet_ntoa*( ).

19874 **RATIONALE**

19875 None.

19876 **FUTURE DIRECTIONS**

19877 None.

19878 **SEE ALSO**

19879 *endhostent*( ), *endnetent*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<arpa/inet.h>**

19880 **CHANGE HISTORY**

19881 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

inet_ntop, inet_pton — convert IPv4 and IPv6 addresses between binary and text form

SYNOPSIS

```
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *restrict src,
    char *restrict dst, socklen_t size);
int inet_pton(int af, const char *restrict src, void *restrict dst);
```

DESCRIPTION

The *inet_ntop*( ) function shall convert a numeric address into a text string suitable for IP6 presentation. The *af* argument shall specify the family of the address. This can be AF_INET or AF_INET6. The *src* argument points to a buffer holding an IPv4 address if the *af* argument is IP6 AF_INET, or an IPv6 address if the *af* argument is AF_INET6; the address must be in network byte order. The *dst* argument points to a buffer where the function stores the resulting text string; it shall not be NULL. The *size* argument specifies the size of this buffer, which shall be IP6 large enough to hold the text string (INET_ADDRSTRLEN characters for IPv4, INET6_ADDRSTRLEN characters for IPv6).

The *inet_pton*( ) function shall convert an address in its standard text presentation form into its IP6 numeric binary form. The *af* argument shall specify the family of the address. The AF_INET and AF_INET6 address families shall be supported. The *src* argument points to the string being passed in. The *dst* argument points to a buffer into which the function stores the numeric IP6 address; this shall be large enough to hold the numeric address (32 bits for AF_INET, 128 bits for AF_INET6).

If the *af* argument of *inet_pton*( ) is AF_INET, the *src* string shall be in the standard IPv4 dotted-decimal form:

```
ddd.ddd.ddd.ddd
```

where "ddd" is a one to three digit decimal number between 0 and 255 (see *inet_addr*( )). The *inet_pton*( ) function does not accept other formats (such as the octal numbers, hexadecimal numbers, and fewer than four numbers that *inet_addr*( ) accepts).

IP6 If the *af* argument of *inet_pton*( ) is AF_INET6, the *src* string shall be in one of the following standard IPv6 text forms:

1. The preferred form is "x:x:x:x:x:x:x:x", where the 'x's are the hexadecimal values of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted, but there shall be at least one numeral in every field.

2. A string of contiguous zero fields in the preferred form can be shown as "::". The "::" can only appear once in an address. Unspecified addresses ("0:0:0:0:0:0:0:0") may be represented simply as "::".

3. A third form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is "x:x:x:x:x:x:d.d.d.d", where the 'x's are the hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

Note: A more extensive description of the standard representations of IPv6 addresses can be found in RFC 2373.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

RETURN VALUE

The *inet_ntop*( ) function shall return a pointer to the buffer containing the text string if the conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

The *inet_pton*( ) function shall return 1 if the conversion succeeds, with the address pointed to by *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or a valid IPv6 address string, or –1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is unknown.

IP6

ERRORS

The *inet_ntop*( ) and *inet_pton*( ) functions shall fail if:

[EAFNOSUPPORT]
The *af* argument is invalid.

[ENOSPC] The size of the *inet_ntop*( ) result buffer is inadequate.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<arpa/inet.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IPv6 extensions are marked.

The **restrict** keyword is added to the *inet_ntop*( ) and *inet_pton*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding ‘‘the address must be in network byte order’’ to the end of the fourth sentence of the first paragraph in the DESCRIPTION.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

initstate, random, setstate, srandom — pseudo-random number functions

SYNOPSIS

XSI

```
#include <stdlib.h>

char *initstate(unsigned seed, char *state, size_t size);
long random(void);
char *setstate(const char *state);
void srandom(unsigned seed);
```

DESCRIPTION

The *random*( ) function shall use a non-linear additive feedback random-number generator employing a default state array size of 31 **long** integers to return successive pseudo-random numbers in the range from 0 to 2^{31}–1. The period of this random-number generator is approximately 16 x (2^{31}–1). The size of the state array determines the period of the random-number generator. Increasing the state array size shall increase the period.

With 256 bytes of state information, the period of the random-number generator shall be greater than 2^{69}.

Like *rand*( ), *random*( ) shall produce by default a sequence of numbers that can be duplicated by calling *srandom*( ) with 1 as the seed.

The *srandom*( ) function shall initialize the current state array using the value of *seed*.

The *initstate*( ) and *setstate*( ) functions handle restarting and changing random-number generators. The *initstate*( ) function allows a state array, pointed to by the *state* argument, to be initialized for future use. The *size* argument, which specifies the size in bytes of the state array, shall be used by *initstate*( ) to decide what type of random-number generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one of these values. If *initstate*( ) is called with $8 \le size < 32$, then *random*( ) shall use a simple linear congruential random number generator. The *seed* argument specifies a starting point for the random-number sequence and provides for restarting at the same point. The *initstate*( ) function shall return a pointer to the previous state information array.

If *initstate*( ) has not been called, then *random*( ) shall behave as though *initstate*( ) had been called with *seed*=1 and *size*=128.

Once a state has been initialized, *setstate*( ) allows switching between state arrays. The array defined by the *state* argument shall be used for further random-number generation until *initstate*( ) is called or *setstate*( ) is called again. The *setstate*( ) function shall return a pointer to the previous state array.

RETURN VALUE

If *initstate*( ) is called with *size* less than 8, it shall return NULL.

The *random*( ) function shall return the generated pseudo-random number.

The *srandom*( ) function shall not return a value.

Upon successful completion, *initstate*( ) and *setstate*( ) shall return a pointer to the previous state array; otherwise, a null pointer shall be returned.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19998 **ERRORS**

19999 No errors are defined.

20000 **EXAMPLES**

20001 None.

20002 **APPLICATION USAGE**

20003 After initialization, a state array can be restarted at a different point in one of two ways:

20004 1. The *initstate*( ) function can be used, with the desired seed, state array, and size of the
20005 array.

20006 2. The *setstate*( ) function, with the desired state, can be used, followed by *srandom*( ) with the
20007 desired seed. The advantage of using both of these functions is that the size of the state
20008 array does not have to be saved once it is initialized.

20009 Although some implementations of *random*( ) have written messages to standard error, such
20010 implementations do not conform to IEEE Std 1003.1-2001.

20011 Issue 5 restored the historical behavior of this function.

20012 Threaded applications should use *erand48*( ), *nrand48*( ), or *jrand48*( ) instead of *random*( ) when |
20013 an independent random number sequence in multiple threads is required.

20014 **RATIONALE**

20015 None.

20016 **FUTURE DIRECTIONS**

20017 None.

20018 **SEE ALSO**

20019 *drand48*( ), *rand*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

20020 **CHANGE HISTORY**

20021 First released in Issue 4, Version 2.

20022 **Issue 5**

20023 Moved from X/OPEN UNIX extension to BASE.

20024 In the DESCRIPTION, the phrase ‘‘values smaller than 8’’ is replaced with ‘‘values greater than
20025 or equal to 8, or less than 32’’, ‘‘*size*<8’’ is replaced with ‘‘8≤*size* <32’’, and a new first paragraph
20026 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE
20027 indicating that these changes restore the historical behavior of the function.

20028 **Issue 6**

20029 In the DESCRIPTION, duplicate text ‘‘For values greater than or equal to 8 . . .’’ is removed.

20030 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand_r*( ) from the |
20031 list of suggested functions in the APPLICATION USAGE section. |

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20032 **NAME**

20033 insque, remque — insert or remove an element in a queue

20034 **SYNOPSIS**

20035 XSI
```
#include <search.h>

void insque(void *element, void *pred);
void remque(void *element);
```
20038

20039 **DESCRIPTION**

20040 The *insque*( ) and *remque*( ) functions shall manipulate queues built from doubly-linked lists. The
20041 queue can be either circular or linear. An application using *insque*( ) or *remque*( ) shall ensure it
20042 defines a structure in which the first two members of the structure are pointers to the same type
20043 of structure, and any further members are application-specific. The first member of the structure
20044 is a forward pointer to the next entry in the queue. The second member is a backward pointer to
20045 the previous entry in the queue. If the queue is linear, the queue is terminated with null
20046 pointers. The names of the structure and of the pointer members are not subject to any special
20047 restriction.

20048 The *insque*( ) function shall insert the element pointed to by *element* into a queue immediately
20049 after the element pointed to by *pred*.

20050 The *remque*( ) function shall remove the element pointed to by *element* from a queue.

20051 If the queue is to be used as a linear list, invoking *insque*(&*element*, NULL), where *element* is the
20052 initial element of the queue, shall initialize the forward and backward pointers of *element* to null
20053 pointers.

20054 If the queue is to be used as a circular list, the application shall ensure it initializes the forward
20055 pointer and the backward pointer of the initial element of the queue to the element's own
20056 address.

20057 **RETURN VALUE**

20058 The *insque*( ) and *remque*( ) functions do not return a value.

20059 **ERRORS**

20060 No errors are defined.

20061 **EXAMPLES**

20062 **Creating a Linear Linked List**

20063 The following example creates a linear linked list.

```
20064 #include <search.h>
20065 ...
20066 struct myque element1;
20067 struct myque element2;

20068 char *data1 = "DATA1";
20069 char *data2 = "DATA2";
20070 ...
20071 element1.data = data1;
20072 element2.data = data2;

20073 insque (&element1, NULL);
20074 insque (&element2, &element1);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20075 **Creating a Circular Linked List**

20076 The following example creates a circular linked list.

```
20077 #include <search.h>
20078 ...
20079 struct myque element1;
20080 struct myque element2;

20081 char *data1 = "DATA1";
20082 char *data2 = "DATA2";
20083 ...
20084 element1.data = data1;
20085 element2.data = data2;

20086 element1.fwd = &element1;
20087 element1.bck = &element1;

20088 insque (&element2, &element1);
```

20089 **Removing an Element**

20090 The following example removes the element pointed to by *element1*.

```
20091 #include <search.h>
20092 ...
20093 struct myque element1;
20094 ...
20095 remque (&element1);
```

20096 **APPLICATION USAGE**

20097 The historical implementations of these functions described the arguments as being of type
20098 **struct qelem *** rather than as being of type **void *** as defined here. In those implementations,
20099 **struct qelem** was commonly defined in **<search.h>** as:

```
20100 struct qelem {
20101     struct qelem  *q_forw;
20102     struct qelem  *q_back;
20103 };
```

20104 Applications using these functions, however, were never able to use this structure directly since
20105 it provided no room for the actual data contained in the elements. Most applications defined
20106 structures that contained the two pointers as the initial elements and also provided space for, or
20107 pointers to, the object's data. Applications that used these functions to update more than one
20108 type of table also had the problem of specifying two or more different structures with the same
20109 name, if they literally used **struct qelem** as specified.

20110 As described here, the implementations were actually expecting a structure type where the first
20111 two members were forward and backward pointers to structures. With C compilers that didn't
20112 provide function prototypes, applications used structures as specified in the DESCRIPTION
20113 above and the compiler did what the application expected.

20114 If this method had been carried forward with an ISO C standard compiler and the historical
20115 function prototype, most applications would have to be modified to cast pointers to the
20116 structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By
20117 specifying **void *** as the argument type, applications do not need to change (unless they
20118 specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20119 **RATIONALE**

20120 None.

20121 **FUTURE DIRECTIONS**

20122 None.

20123 **SEE ALSO**

20124 The Base Definitions volume of IEEE Std 1003.1-2001, **<search.h>**

20125 **CHANGE HISTORY**

20126 First released in Issue 4, Version 2.

20127 **Issue 5**

20128 Moved from X/OPEN UNIX extension to BASE.

20129 **Issue 6**

20130 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20131 **NAME**

20132 ioctl — control a STREAMS device (**STREAMS**)

20133 **SYNOPSIS**

20134 XSR
```
#include <stropts.h>
```

20135
```
int ioctl(int fildes, int request, ... /* arg */);
```

20136

20137 **DESCRIPTION**

20138 The *ioctl*( ) function shall perform a variety of control functions on STREAMS devices. For non-
20139 STREAMS devices, the functions performed by this call are unspecified. The *request* argument
20140 and an optional third argument (with varying type) shall be passed to and interpreted by the
20141 appropriate part of the STREAM associated with *fildes*.

20142 The *fildes* argument is an open file descriptor that refers to a device.

20143 The *request* argument selects the control function to be performed and shall depend on the
20144 STREAMS device being addressed.

20145 The *arg* argument represents additional information that is needed by this specific STREAMS
20146 device to perform the requested function. The type of *arg* depends upon the particular control
20147 request, but it shall be either an integer or a pointer to a device-specific data structure.

20148 The *ioctl*( ) commands applicable to STREAMS, their arguments, and error conditions that apply
20149 to each individual command are described below.

20150 The following *ioctl*( ) commands, with error values indicated, are applicable to all STREAMS
20151 files:

20152 I_PUSH Pushes the module whose name is pointed to by *arg* onto the top of the
20153 current STREAM, just below the STREAM head. It then calls the *open*( )
20154 function of the newly-pushed module.

20155 The *ioctl*( ) function with the I_PUSH command shall fail if:

20156 [EINVAL] Invalid module name.

20157 [ENXIO] Open function of new module failed.

20158 [ENXIO] Hangup received on *fildes*.

20159 I_POP Removes the module just below the STREAM head of the STREAM pointed to
20160 by *fildes*. The *arg* argument should be 0 in an I_POP request.

20161 The *ioctl*( ) function with the I_POP command shall fail if:

20162 [EINVAL] No module present in the STREAM.

20163 [ENXIO] Hangup received on *fildes*.

20164 I_LOOK Retrieves the name of the module just below the STREAM head of the
20165 STREAM pointed to by *fildes*, and places it in a character string pointed to by
20166 *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,
20167 where FMNAMESZ is defined in **<stropts.h>**.

20168 The *ioctl*( ) function with the I_LOOK command shall fail if:

20169 [EINVAL] No module present in the STREAM.

20170 I_FLUSH Flushes read and/or write queues, depending on the value of *arg*. Valid *arg*
20171 values are:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20172 FLUSHR Flush all read queues.

20173 FLUSHW Flush all write queues.

20174 FLUSHRW Flush all read and all write queues.

20175 The *ioctl*( ) function with the I_FLUSH command shall fail if:

20176 [EINVAL] Invalid *arg* value.

20177 [EAGAIN] or [ENOSR]
20178 Unable to allocate buffers for flush message.

20179 [ENXIO] Hangup received on *fildes*.

20180 I_FLUSHBAND Flushes a particular band of messages. The *arg* argument points to a **bandinfo**
20181 structure. The *bi_flag* member may be one of FLUSHR, FLUSHW, or
20182 FLUSHRW as described above. The *bi_pri* member determines the priority
20183 band to be flushed.

20184 I_SETSIG Requests that the STREAMS implementation send the SIGPOLL signal to the
20185 calling process when a particular event has occurred on the STREAM
20186 associated with *fildes*. I_SETSIG supports an asynchronous processing
20187 capability in STREAMS. The value of *arg* is a bitmask that specifies the events
20188 for which the process should be signaled. It is the bitwise-inclusive OR of any
20189 combination of the following constants:

20190 S_RDNORM A normal (priority band set to 0) message has arrived at the
20191 head of a STREAM head read queue. A signal shall be
20192 generated even if the message is of zero length.

20193 S_RDBAND A message with a non-zero priority band has arrived at the
20194 head of a STREAM head read queue. A signal shall be
20195 generated even if the message is of zero length.

20196 S_INPUT A message, other than a high-priority message, has arrived
20197 at the head of a STREAM head read queue. A signal shall be
20198 generated even if the message is of zero length.

20199 S_HIPRI A high-priority message is present on a STREAM head read
20200 queue. A signal shall be generated even if the message is of
20201 zero length.

20202 S_OUTPUT The write queue for normal data (priority band 0) just
20203 below the STREAM head is no longer full. This notifies the
20204 process that there is room on the queue for sending (or
20205 writing) normal data downstream.

20206 S_WRNORM Equivalent to S_OUTPUT.

20207 S_WRBAND The write queue for a non-zero priority band just below the
20208 STREAM head is no longer full. This notifies the process
20209 that there is room on the queue for sending (or writing)
20210 priority data downstream.

20211 S_MSG A STREAMS signal message that contains the SIGPOLL
20212 signal has reached the front of the STREAM head read
20213 queue.

20214 S_ERROR Notification of an error condition has reached the STREAM
20215 head.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

S_HANGUP Notification of a hangup has reached the STREAM head.

S_BANDURG When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.

If *arg* is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with *fildes*.

Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.

The *ioctl*( ) function with the I_SETSIG command shall fail if:

[EINVAL] The value of *arg* is invalid.

[EINVAL] The value of *arg* is 0 and the calling process is not registered to receive the SIGPOLL signal.

[EAGAIN] There were insufficient resources to store the signal request.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an **int** pointed to by *arg*, where the events are those specified in the description of I_SETSIG above.

The *ioctl*( ) function with the I_GETSIG command shall fail if:

[EINVAL] Process is not registered to receive the SIGPOLL signal.

I_FIND Compares the names of all modules currently present in the STREAM to the name pointed to by *arg*, and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.

The *ioctl*( ) function with the I_FIND command shall fail if:

[EINVAL] *arg* does not contain a valid module name.

I_PEEK Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to *getmsg*( ) except that this command does not remove the message from the queue. The *arg* argument points to a **strpeek** structure.

The application shall ensure that the *maxlen* member in the **ctlbuf** and **databuf** **strbuf** structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The *flags* member may be marked RS_HIPRI or 0, as described by *getmsg*( ). If the process sets *flags* to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in *flags* and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, **ctlbuf** specifies information in the control buffer, **databuf** specifies information in the data buffer, and *flags* contains the value RS_HIPRI or 0.

I_SRDOPT Sets the read mode using the value of the argument *arg*. Read modes are described in *read*( ). Valid *arg* flags are:

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RNORM Byte-stream mode, the default.

RMSGD Message-discard mode.

RMSGN Message-nondiscard mode.

The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in the other flag overriding RNORM which is the default.

In addition, treatment of control messages by the STREAM head may be changed by setting any of the following flags in *arg*:

RPROTNORM Fail *read*( ) with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.

RPROTDAT Deliver the control part of a message as data when a process issues a *read*( ).

RPROTDIS Discard the control part of a message, delivering any data portion, when a process issues a *read*( ).

The *ioctl*( ) function with the I_SRDOPT command shall fail if:

[EINVAL] The *arg* argument is not valid.

I_GRDOPT Returns the current read mode setting, as described above, in an **int** pointed to by the argument *arg*. Read modes are described in *read*( ).

I_NREAD Counts the number of data bytes in the data part of the first message on the STREAM head read queue and places this value in the **int** pointed to by *arg*. The return value for the command shall be the number of messages on the STREAM head read queue. For example, if 0 is returned in *arg*, but the *ioctl*( ) return value is greater than 0, this indicates that a zero-length message is next on the queue.

I_FDINSERT Creates a message from specified buffer(s), adds information about another STREAM, and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The *arg* argument points to a **strfdinsert** structure.

The application shall ensure that the *len* member in the **ctlbuf strbuf** structure is set to the size of a **t_uscalar_t** plus the number of bytes of control information to be sent with the message. The *fildes* member specifies the file descriptor of the other STREAM, and the *offset* member, which must be suitably aligned for use as a **t_uscalar_t**, specifies the offset from the start of the control buffer where I_FDINSERT shall store a **t_uscalar_t** whose interpretation is specific to the STREAM end. The application shall ensure that the *len* member in the **databuf strbuf** structure is set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.

The *flags* member specifies the type of message to be created. A normal message is created if *flags* is set to 0, and a high-priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if the STREAM write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

and O_NONBLOCK is set. Instead, it fails and sets *errno* to [EAGAIN].

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the STREAM, regardless of priority or whether O_NONBLOCK has been specified. No partial message is sent.

The *ioctl*( ) function with the I_FDINSERT command shall fail if:

[EAGAIN] A non-priority message is specified, the O_NONBLOCK flag is set, and the STREAM write queue is full due to internal flow control conditions.

[EAGAIN] or [ENOSR]
Buffers cannot be allocated for the message that is to be created.

[EINVAL] One of the following:

— The *fildes* member of the **strfdinsert** structure is not a valid, open STREAM file descriptor.

— The size of a **t_uscalar_t** plus *offset* is greater than the *len* member for the buffer specified through **ctlbuf**.

— The *offset* member does not specify a properly-aligned location in the data buffer.

— An undefined value is stored in *flags*.

[ENXIO] Hangup received on the STREAM identified by either the *fildes* argument or the *fildes* member of the **strfdinsert** structure.

[ERANGE] The *len* member for the buffer specified through **databuf** does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module; or the *len* member for the buffer specified through **databuf** is larger than the maximum configured size of the data part of a message; or the *len* member for the buffer specified through **ctlbuf** is larger than the maximum configured size of the control part of a message.

I_STR Constructs an internal STREAMS *ioctl*( ) message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send *ioctl*( ) requests to downstream modules and drivers. It allows information to be sent with *ioctl*( ), and returns to the process any information sent upstream by the downstream recipient. I_STR shall block until the system responds with either a positive or negative acknowledgement message, or until the request times out after some period of time. If the request times out, it shall fail with *errno* set to [ETIME].

At most, one I_STR can be active on a STREAM. Further I_STR calls shall block until the active I_STR completes at the STREAM head. The default timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has no effect on this call.

To send requests downstream, the application shall ensure that *arg* points to a **strioctl** structure.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *ic_cmd* member is the internal *ioctl*() command intended for a downstream module or driver and *ic_timout* is the number of seconds (–1=infinite, 0=use implementation-defined timeout interval, >0=as specified) an I_STR request shall wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument, and *ic_dp* is a pointer to the data argument. The *ic_len* member has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the process (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the STREAM can return).

The STREAM head shall convert the information pointed to by the **strioctl** structure to an internal *ioctl*() command message and send it downstream.

The *ioctl*() function with the I_STR command shall fail if:

[EAGAIN] or [ENOSR]
: Unable to allocate buffers for the *ioctl*() message.

[EINVAL]
: The *ic_len* member is less than 0 or larger than the maximum configured size of the data part of a message, or *ic_timout* is less than –1.

[ENXIO]
: Hangup received on *fildes*.

[ETIME]
: A downstream *ioctl*() timed out before acknowledgement was received.

An I_STR can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the STREAM head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the *ioctl*() command sent downstream fails. For these cases, I_STR shall fail with *errno* set to the value in the message.

I_SWROPT
: Sets the write mode using the value of the argument *arg*. Valid bit settings for *arg* are:

  SNDZERO
  : Send a zero-length message downstream when a *write*() of 0 bytes occurs. To not send a zero-length message when a *write*() of 0 bytes occurs, the application shall ensure that this bit is not set in *arg* (for example, *arg* would be set to 0).

  The *ioctl*() function with the I_SWROPT command shall fail if:

  [EINVAL]
  : *arg* is not the above value.

I_GWROPT
: Returns the current write mode setting, as described above, in the **int** that is pointed to by the argument *arg*.

I_SENDFD
: Creates a new reference to the open file description associated with the file descriptor *arg*, and writes a message on the STREAMS-based pipe *fildes* containing this reference, together with the user ID and group ID of the calling process.

  The *ioctl*() function with the I_SENDFD command shall fail if:

  [EAGAIN]
  : The sending STREAM is unable to allocate a message block to contain the file pointer; or the read queue of the receiving STREAM head is full and cannot accept the message sent by I_SENDFD.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EBADF] The *arg* argument is not a valid, open file descriptor.

[EINVAL] The *fildes* argument is not connected to a STREAM pipe.

[ENXIO] Hangup received on *fildes*.

I_RECVFD Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The *arg* argument is a pointer to a **strrecvfd** data structure as defined in **<stropts.h>**.

The *fd* member is a file descriptor. The *uid* and *gid* members are the effective user ID and effective group ID, respectively, of the sending process.

If O_NONBLOCK is not set, I_RECVFD shall block until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD shall fail with *errno* set to [EAGAIN] if no message is present at the STREAM head.

If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor shall be allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the *fd* member of the **strrecvfd** structure pointed to by *arg*.

The *ioctl*( ) function with the I_RECVFD command shall fail if:

[EAGAIN] A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.

[EBADMSG] The message at the STREAM head read queue is not a message containing a passed file descriptor.

[EMFILE] The process has the maximum number of file descriptors currently open that it is allowed.

[ENXIO] Hangup received on *fildes*.

I_LIST Allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If *arg* is a null pointer, the return value shall be the number of modules, including the driver, that are on the STREAM pointed to by *fildes*. This lets the process allocate enough space for the module names. Otherwise, it should point to a **str_list** structure.

The *sl_nmods* member indicates the number of entries the process has allocated in the array. Upon return, the *sl_modlist* member of the **str_list** structure shall contain the list of module names, and the number of entries that have been filled into the *sl_modlist* array is found in the *sl_nmods* member (the number includes the number of modules including the driver). The return value from *ioctl*( ) shall be 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules (*sl_nmods*) is satisfied.

The *ioctl*( ) function with the I_LIST command shall fail if:

[EINVAL] The *sl_nmods* member is less than 1.

[EAGAIN] or [ENOSR]
Unable to allocate buffers.

I_ATMARK Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The *arg* argument determines

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

how the checking is done when there may be multiple marked messages on the STREAM head read queue. It may take on the following values:

ANYMARK Check if the message is marked.

LASTMARK Check if the message is the last one marked on the queue.

The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is permitted.

The return value shall be 1 if the mark condition is satisfied; otherwise, the value shall be 0.

The *ioctl*( ) function with the I_ATMARK command shall fail if:

[EINVAL] Invalid *arg* value.

I_CKBAND Checks if the message of a given priority band exists on the STREAM head read queue. This shall return 1 if a message of the given priority exists, 0 if no such message exists, or –1 on error. *arg* should be of type **int**.

The *ioctl*( ) function with the I_CKBAND command shall fail if:

[EINVAL] Invalid *arg* value.

I_GETBAND Returns the priority band of the first message on the STREAM head read queue in the integer referenced by *arg*.

The *ioctl*( ) function with the I_GETBAND command shall fail if:

[ENODATA] No message on the STREAM head read queue.

I_CANPUT Checks if a certain band is writable. *arg* is set to the priority band in question. The return value shall be 0 if the band is flow-controlled, 1 if the band is writable, or –1 on error.

The *ioctl*( ) function with the I_CANPUT command shall fail if:

[EINVAL] Invalid *arg* value.

I_SETCLTIME This request allows the process to set the time the STREAM head shall delay when a STREAM is closing and there is data on the write queues. Before closing each module or driver, if there is data on its write queue, the STREAM head shall delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, it shall be flushed. The *arg* argument is a pointer to an integer specifying the number of milliseconds to delay, rounded up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM, an implementation-defined default timeout interval is used.

The *ioctl*( ) function with the I_SETCLTIME command shall fail if:

[EINVAL] Invalid *arg* value.

I_GETCLTIME Returns the close time delay in the integer pointed to by *arg*.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20474 **Multiplexed STREAMS Configurations**

20475 The following commands are used for connecting and disconnecting multiplexed STREAMS
20476 configurations. These commands use an implementation-defined default timeout interval.

20477 I_LINK Connects two STREAMs, where *fildes* is the file descriptor of the STREAM
20478 connected to the multiplexing driver, and *arg* is the file descriptor of the
20479 STREAM connected to another driver. The STREAM designated by *arg* is
20480 connected below the multiplexing driver. I_LINK requires the multiplexing
20481 driver to send an acknowledgement message to the STREAM head regarding
20482 the connection. This call shall return a multiplexer ID number (an identifier
20483 used to disconnect the multiplexer; see I_UNLINK) on success, and –1 on
20484 failure.

20485 The *ioctl*( ) function with the I_LINK command shall fail if:

20486 [ENXIO] Hangup received on *fildes*.

20487 [ETIME] Timeout before acknowledgement message was received at
20488 STREAM head.

20489 [EAGAIN] or [ENOSR]
20490 Unable to allocate STREAMS storage to perform the
20491 I_LINK.

20492 [EBADF] The *arg* argument is not a valid, open file descriptor.

20493 [EINVAL] The *fildes* argument does not support multiplexing; or *arg* is
20494 not a STREAM or is already connected downstream from a
20495 multiplexer; or the specified I_LINK operation would
20496 connect the STREAM head in more than one place in the
20497 multiplexed STREAM.

20498 An I_LINK can also fail while waiting for the multiplexing driver to
20499 acknowledge the request, if a message indicating an error or a hangup is
20500 received at the STREAM head of *fildes*. In addition, an error code can be
20501 returned in the positive or negative acknowledgement message. For these
20502 cases, I_LINK fails with *errno* set to the value in the message.

20503 I_UNLINK Disconnects the two STREAMs specified by *fildes* and *arg*. *fildes* is the file
20504 descriptor of the STREAM connected to the multiplexing driver. The *arg*
20505 argument is the multiplexer ID number that was returned by the I_LINK
20506 *ioctl*( ) command when a STREAM was connected downstream from the
20507 multiplexing driver. If *arg* is MUXID_ALL, then all STREAMs that were
20508 connected to *fildes* shall be disconnected. As in I_LINK, this command
20509 requires acknowledgement.

20510 The *ioctl*( ) function with the I_UNLINK command shall fail if:

20511 [ENXIO] Hangup received on *fildes*.

20512 [ETIME] Timeout before acknowledgement message was received at
20513 STREAM head.

20514 [EAGAIN] or [ENOSR]
20515 Unable to allocate buffers for the acknowledgement
20516 message.

20517 [EINVAL] Invalid multiplexer ID number.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20518 An I_UNLINK can also fail while waiting for the multiplexing driver to
20519 acknowledge the request if a message indicating an error or a hangup is
20520 received at the STREAM head of *fildes*. In addition, an error code can be
20521 returned in the positive or negative acknowledgement message. For these
20522 cases, I_UNLINK shall fail with *errno* set to the value in the message.

20523 I_PLINK Creates a *persistent connection* between two STREAMs, where *fildes* is the file
20524 descriptor of the STREAM connected to the multiplexing driver, and *arg* is the
20525 file descriptor of the STREAM connected to another driver. This call shall
20526 create a persistent connection which can exist even if the file descriptor *fildes*
20527 associated with the upper STREAM to the multiplexing driver is closed. The
20528 STREAM designated by *arg* gets connected via a persistent connection below
20529 the multiplexing driver. I_PLINK requires the multiplexing driver to send an
20530 acknowledgement message to the STREAM head. This call shall return a
20531 multiplexer ID number (an identifier that may be used to disconnect the
20532 multiplexer; see I_PUNLINK) on success, and –1 on failure.

20533 The *ioctl*( ) function with the I_PLINK command shall fail if:

20534 [ENXIO] Hangup received on *fildes*.

20535 [ETIME] Timeout before acknowledgement message was received at
20536 STREAM head.

20537 [EAGAIN] or [ENOSR]
20538 Unable to allocate STREAMS storage to perform the
20539 I_PLINK.

20540 [EBADF] The *arg* argument is not a valid, open file descriptor.

20541 [EINVAL] The *fildes* argument does not support multiplexing; or *arg* is
20542 not a STREAM or is already connected downstream from a
20543 multiplexer; or the specified I_PLINK operation would
20544 connect the STREAM head in more than one place in the
20545 multiplexed STREAM.

20546 An I_PLINK can also fail while waiting for the multiplexing driver to
20547 acknowledge the request, if a message indicating an error or a hangup is
20548 received at the STREAM head of *fildes*. In addition, an error code can be
20549 returned in the positive or negative acknowledgement message. For these
20550 cases, I_PLINK shall fail with *errno* set to the value in the message.

20551 I_PUNLINK Disconnects the two STREAMs specified by *fildes* and *arg* from a persistent
20552 connection. The *fildes* argument is the file descriptor of the STREAM
20553 connected to the multiplexing driver. The *arg* argument is the multiplexer ID
20554 number that was returned by the I_PLINK *ioctl*( ) command when a STREAM
20555 was connected downstream from the multiplexing driver. If *arg* is
20556 MUXID_ALL, then all STREAMs which are persistent connections to *fildes*
20557 shall be disconnected. As in I_PLINK, this command requires the multiplexing
20558 driver to acknowledge the request.

20559 The *ioctl*( ) function with the I_PUNLINK command shall fail if:

20560 [ENXIO] Hangup received on *fildes*.

20561 [ETIME] Timeout before acknowledgement message was received at
20562 STREAM head.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EAGAIN] or [ENOSR]
Unable to allocate buffers for the acknowledgement message.

[EINVAL] Invalid multiplexer ID number.

An I_PUNLINK can also fail while waiting for the multiplexing driver to acknowledge the request if a message indicating an error or a hangup is received at the STREAM head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_PUNLINK shall fail with *errno* set to the value in the message.

RETURN VALUE

Upon successful completion, *ioctl*( ) shall return a value other than –1 that depends upon the STREAMS device control function. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

Under the following general conditions, *ioctl*( ) shall fail if:

[EBADF] The *fildes* argument is not a valid open file descriptor.

[EINTR] A signal was caught during the *ioctl*( ) operation.

[EINVAL] The STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

If an underlying device driver detects an error, then *ioctl*( ) shall fail if:

[EINVAL] The *request* or *arg* argument is not valid for this device.

[EIO] Some physical I/O error has occurred.

[ENOTTY] The *fildes* argument is not associated with a STREAMS device that accepts control functions.

[ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service requested cannot be performed on this particular sub-device.

[ENODEV] The *fildes* argument refers to a valid STREAMS device, but the corresponding device driver does not support the *ioctl*( ) function.

If a STREAM is connected downstream from a multiplexer, any *ioctl*( ) command except I_UNLINK and I_PUNLINK shall set *errno* to [EINVAL].

EXAMPLES

None.

APPLICATION USAGE

The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 38), *close*( ), *fcntl*( ), *getmsg*( ), *open*( ), *pipe*( ), *poll*( ), *putmsg*( ), *read*( ), *sigaction*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stropts.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20604 **CHANGE HISTORY**

20605 First released in Issue 4, Version 2.

20606 **Issue 5**

20607 Moved from X/OPEN UNIX extension to BASE.

20608 **Issue 6**

20609 The Open Group Corrigendum U028/4 is applied, correcting text in the I_FDINSERT [EINVAL]
20610 case to refer to *ctlbuf*.

20611 This function is marked as part of the XSI STREAMS Option Group.

20612 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

20613 **NAME**

20614 isalnum — test for an alphanumeric character

20615 **SYNOPSIS**

20616
```
#include <ctype.h>
```

20617
```
int isalnum(int c);
```

20618 **DESCRIPTION**

20619 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20620 conflict between the requirements described here and the ISO C standard is unintentional. This
20621 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20622 The *isalnum*( ) function shall test whether *c* is a character of class **alpha** or **digit** in the program's
20623 current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20624 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
20625 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
20626 behavior is undefined.

20627 **RETURN VALUE**

20628 The *isalnum*( ) function shall return non-zero if *c* is an alphanumeric character; otherwise, it shall
20629 return 0.

20630 **ERRORS**

20631 No errors are defined.

20632 **EXAMPLES**

20633 None.

20634 **APPLICATION USAGE**

20635 To ensure applications portability, especially across natural languages, only this function and
20636 those listed in the SEE ALSO section should be used for character classification.

20637 **RATIONALE**

20638 None.

20639 **FUTURE DIRECTIONS**

20640 None.

20641 **SEE ALSO**

20642 *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ),
20643 *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**,
20644 **<stdio.h>**

20645 **CHANGE HISTORY**

20646 First released in Issue 1. Derived from Issue 1 of the SVID.

20647 **Issue 6**

20648 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20649 **NAME**

20650 isalpha — test for an alphabetic character

20651 **SYNOPSIS**

20652
```
#include <ctype.h>
```

20653
```
int isalpha(int c);
```

20654 **DESCRIPTION**

20655 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20656 conflict between the requirements described here and the ISO C standard is unintentional. This
20657 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20658 The *isalpha*( ) function shall test whether *c* is a character of class **alpha** in the program's current
20659 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20660 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
20661 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
20662 behavior is undefined.

20663 **RETURN VALUE**

20664 The *isalpha*( ) function shall return non-zero if *c* is an alphabetic character; otherwise, it shall
20665 return 0.

20666 **ERRORS**

20667 No errors are defined.

20668 **EXAMPLES**

20669 None.

20670 **APPLICATION USAGE**

20671 To ensure applications portability, especially across natural languages, only this function and
20672 those listed in the SEE ALSO section should be used for character classification.

20673 **RATIONALE**

20674 None.

20675 **FUTURE DIRECTIONS**

20676 None.

20677 **SEE ALSO**

20678 *isalnum*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ),
20679 *isxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
20680 **<ctype.h>**, **<stdio.h>**

20681 **CHANGE HISTORY**

20682 First released in Issue 1. Derived from Issue 1 of the SVID.

20683 **Issue 6**

20684 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20685 **NAME**

20686 isascii — test for a 7-bit US-ASCII character

20687 **SYNOPSIS**

20688 XSI
```
#include <ctype.h>
```

20689
```
int isascii(int c);
```

20690

20691 **DESCRIPTION**

20692 The *isascii*( ) function shall test whether *c* is a 7-bit US-ASCII character code.

20693 The *isascii*( ) function is defined on all integer values.

20694 **RETURN VALUE**

20695 The *isascii*( ) function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and
20696 octal 0177 inclusive; otherwise, it shall return 0.

20697 **ERRORS**

20698 No errors are defined.

20699 **EXAMPLES**

20700 None.

20701 **APPLICATION USAGE**

20702 None.

20703 **RATIONALE**

20704 None.

20705 **FUTURE DIRECTIONS**

20706 None.

20707 **SEE ALSO**

20708 The Base Definitions volume of IEEE Std 1003.1-2001, **<ctype.h>**

20709 **CHANGE HISTORY**

20710 First released in Issue 1. Derived from Issue 1 of the SVID.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

-`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20711 **NAME**

20712 isastream — test a file descriptor (**STREAMS**)

20713 **SYNOPSIS**

20714 XSR
```
#include <stropts.h>
```

20715
```
int isastream(int fildes);
```

20716

20717 **DESCRIPTION**

20718 The *isastream*( ) function shall test whether *fildes*, an open file descriptor, is associated with a
20719 STREAMS-based file.

20720 **RETURN VALUE**

20721 Upon successful completion, *isastream*( ) shall return 1 if *fildes* refers to a STREAMS-based file
20722 and 0 if not. Otherwise, *isastream*( ) shall return –1 and set *errno* to indicate the error.

20723 **ERRORS**

20724 The *isastream*( ) function shall fail if:

20725 [EBADF] The *fildes* argument is not a valid open file descriptor.

20726 **EXAMPLES**

20727 None.

20728 **APPLICATION USAGE**

20729 None.

20730 **RATIONALE**

20731 None.

20732 **FUTURE DIRECTIONS**

20733 None.

20734 **SEE ALSO**

20735 The Base Definitions volume of IEEE Std 1003.1-2001, **<stropts.h>**

20736 **CHANGE HISTORY**

20737 First released in Issue 4, Version 2.

20738 **Issue 5**

20739 Moved from X/OPEN UNIX extension to BASE.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

20740 **NAME**

20741 isatty — test for a terminal device

20742 **SYNOPSIS**

20743
```
#include <unistd.h>
```

20744
```
int isatty(int fildes);
```

20745 **DESCRIPTION**

20746 The *isatty*( ) function shall test whether *fildes*, an open file descriptor, is associated with a
20747 terminal device.

20748 **RETURN VALUE**

20749 The *isatty*( ) function shall return 1 if *fildes* is associated with a terminal; otherwise, it shall return
20750 0 and may set *errno* to indicate the error.

20751 **ERRORS**

20752 The *isatty*( ) function may fail if:

20753 [EBADF] The *fildes* argument is not a valid open file descriptor.

20754 [ENOTTY] The *fildes* argument is not associated with a terminal.

20755 **EXAMPLES**

20756 None.

20757 **APPLICATION USAGE**

20758 The *isatty*( ) function does not necessarily indicate that a human being is available for interaction
20759 via *fildes*. It is quite possible that non-terminal devices are connected to the communications
20760 line.

20761 **RATIONALE**

20762 None.

20763 **FUTURE DIRECTIONS**

20764 None.

20765 **SEE ALSO**

20766 The Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

20767 **CHANGE HISTORY**

20768 First released in Issue 1. Derived from Issue 1 of the SVID.

20769 **Issue 6**

20770 The following new requirements on POSIX implementations derive from alignment with the
20771 Single UNIX Specification:

20772 - The optional setting of *errno* to indicate an error is added.

20773 - The [EBADF] and [ENOTTY] optional error conditions are added.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20774 **NAME**

20775 isblank — test for a blank character

20776 **SYNOPSIS**

20777
```
#include <ctype.h>
```

20778
```
int isblank(int c);
```

20779 **DESCRIPTION**

20780 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20781 conflict between the requirements described here and the ISO C standard is unintentional. This
20782 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20783 The *isblank*( ) function shall test whether *c* is a character of class **blank** in the program's current
20784 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20785 The *c* argument is a type **int**, the value of which the application shall ensure is a character
20786 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
20787 any other value, the behavior is undefined.

20788 **RETURN VALUE**

20789 The *isblank*( ) function shall return non-zero if *c* is a <blank>; otherwise, it shall return 0.

20790 **ERRORS**

20791 No errors are defined.

20792 **EXAMPLES**

20793 None.

20794 **APPLICATION USAGE**

20795 To ensure applications portability, especially across natural languages, only this function and
20796 those listed in the SEE ALSO section should be used for character classification.

20797 **RATIONALE**

20798 None.

20799 **FUTURE DIRECTIONS**

20800 None.

20801 **SEE ALSO**

20802 *isalnum*( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ),
20803 *isxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
20804 **<ctype.h>**

20805 **CHANGE HISTORY**

20806 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20807 **NAME**

20808 iscntrl — test for a control character

20809 **SYNOPSIS**

20810
```
#include <ctype.h>
```

20811
```
int iscntrl(int c);
```

20812 **DESCRIPTION**

20813 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20814 conflict between the requirements described here and the ISO C standard is unintentional. This
20815 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20816 The *iscntrl*( ) function shall test whether *c* is a character of class **cntrl** in the program's current
20817 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20818 The *c* argument is a type **int**, the value of which the application shall ensure is a character
20819 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
20820 any other value, the behavior is undefined.

20821 **RETURN VALUE**

20822 The *iscntrl*( ) function shall return non-zero if *c* is a control character; otherwise, it shall return 0.

20823 **ERRORS**

20824 No errors are defined.

20825 **EXAMPLES**

20826 None.

20827 **APPLICATION USAGE**

20828 To ensure applications portability, especially across natural languages, only this function and
20829 those listed in the SEE ALSO section should be used for character classification.

20830 **RATIONALE**

20831 None.

20832 **FUTURE DIRECTIONS**

20833 None.

20834 **SEE ALSO**

20835 *isalnum*( ), *isalpha*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ),
20836 *isxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
20837 **<ctype.h>**

20838 **CHANGE HISTORY**

20839 First released in Issue 1. Derived from Issue 1 of the SVID.

20840 **Issue 6**

20841 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

isdigit — test for a decimal digit

SYNOPSIS

```
#include <ctype.h>

int isdigit(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *isdigit*( ) function shall test whether *c* is a character of class **digit** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isdigit*( ) function shall return non-zero if *c* is a decimal digit; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20876 **NAME**

20877 isfinite — test for finite value

20878 **SYNOPSIS**

20879
```
#include <math.h>
```

20880
```
int isfinite(real-floating x);
```

20881 **DESCRIPTION**

20882 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20883 conflict between the requirements described here and the ISO C standard is unintentional. This
20884 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20885 The *isfinite*( ) macro shall determine whether its argument has a finite value (zero, subnormal, or
20886 normal, and not infinite or NaN). First, an argument represented in a format wider than its
20887 semantic type is converted to its semantic type. Then determination is based on the type of the
20888 argument.

20889 **RETURN VALUE**

20890 The *isfinite*( ) macro shall return a non-zero value if and only if its argument has a finite value.

20891 **ERRORS**

20892 No errors are defined.

20893 **EXAMPLES**

20894 None.

20895 **APPLICATION USAGE**

20896 None.

20897 **RATIONALE**

20898 None.

20899 **FUTURE DIRECTIONS**

20900 None.

20901 **SEE ALSO**

20902 *fpclassify*( ), *isinf*( ), *isnan*( ), *isnormal*( ), *signbit*( ), the Base Definitions volume of
20903 IEEE Std 1003.1-2001 **<math.h>**

20904 **CHANGE HISTORY**

20905 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20906 **NAME**

20907 isgraph — test for a visible character

20908 **SYNOPSIS**

20909
```
#include <ctype.h>
```

20910
```
int isgraph(int c);
```

20911 **DESCRIPTION**

20912 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20913 conflict between the requirements described here and the ISO C standard is unintentional. This
20914 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20915 The *isgraph*( ) function shall test whether *c* is a character of class **graph** in the program's current
20916 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20917 The *c* argument is an **int**, the value of which the application shall ensure is a character
20918 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
20919 any other value, the behavior is undefined.

20920 **RETURN VALUE**

20921 The *isgraph*( ) function shall return non-zero if *c* is a character with a visible representation;
20922 otherwise, it shall return 0.

20923 **ERRORS**

20924 No errors are defined.

20925 **EXAMPLES**

20926 None.

20927 **APPLICATION USAGE**

20928 To ensure applications portability, especially across natural languages, only this function and
20929 those listed in the SEE ALSO section should be used for character classification.

20930 **RATIONALE**

20931 None.

20932 **FUTURE DIRECTIONS**

20933 None.

20934 **SEE ALSO**

20935 *isalnum*( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ),
20936 *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

20937 **CHANGE HISTORY**

20938 First released in Issue 1. Derived from Issue 1 of the SVID.

20939 **Issue 6**

20940 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20941 **NAME**

20942 isgreater — test if x greater than y

20943 **SYNOPSIS**

20944
```
#include <math.h>
```

20945
```
int isgreater(real-floating x, real-floating y);
```

20946 **DESCRIPTION**

20947 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20948 conflict between the requirements described here and the ISO C standard is unintentional. This
20949 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20950 The *isgreater*( ) macro shall determine whether its first argument is greater than its second
20951 argument. The value of *isgreater*(*x*, *y*) shall be equal to (*x*) > (*y*); however, unlike (*x*) > (*y*),
20952 *isgreater*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

20953 **RETURN VALUE**

20954 Upon successful completion, the *isgreater*( ) macro shall return the value of (*x*) > (*y*).

20955 If *x* or *y* is NaN, 0 shall be returned.

20956 **ERRORS**

20957 No errors are defined.

20958 **EXAMPLES**

20959 None.

20960 **APPLICATION USAGE**

20961 The relational and equality operators support the usual mathematical relationships between
20962 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
20963 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
20964 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
20965 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
20966 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
20967 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
20968 indicates that the argument shall be an expression of **real-floating** type.

20969 **RATIONALE**

20970 None.

20971 **FUTURE DIRECTIONS**

20972 None.

20973 **SEE ALSO**

20974 *isgreaterequal*( ), *isless*( ), *islessequal*( ), *islessgreater*( ), *isunordered*( ), the Base Definitions volume of
20975 IEEE Std 1003.1-2001 **<math.h>**

20976 **CHANGE HISTORY**

20977 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20978 **NAME**

20979 isgreaterequal — test if x is greater than or equal to y

20980 **SYNOPSIS**

20981 `#include <math.h>`

20982 `int isgreaterequal(real-floating x, real-floating y);`

20983 **DESCRIPTION**

20984 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20985 conflict between the requirements described here and the ISO C standard is unintentional. This
20986 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20987 The *isgreaterequal*( ) macro shall determine whether its first argument is greater than or equal to
20988 its second argument. The value of *isgreaterequal*(*x*, *y*) shall be equal to (*x*) >= (*y*); however, unlike
20989 (*x*) >= (*y*), *isgreaterequal*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are
20990 unordered.

20991 **RETURN VALUE**

20992 Upon successful completion, the *isgreaterequal*( ) macro shall return the value of (*x*) >= (*y*).

20993 If *x* or *y* is NaN, 0 shall be returned.

20994 **ERRORS**

20995 No errors are defined.

20996 **EXAMPLES**

20997 None.

20998 **APPLICATION USAGE**

20999 The relational and equality operators support the usual mathematical relationships between
21000 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21001 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21002 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21003 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21004 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21005 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21006 indicates that the argument shall be an expression of **real-floating** type.

21007 **RATIONALE**

21008 None.

21009 **FUTURE DIRECTIONS**

21010 None.

21011 **SEE ALSO**

21012 *isgreater*( ), *isless*( ), *islessequal*( ), *islessgreater*( ), *isunordered*( ), the Base Definitions volume of
21013 IEEE Std 1003.1-2001 **<math.h>**

21014 **CHANGE HISTORY**

21015 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

isinf — test for infinity

SYNOPSIS

```
#include <math.h>

int isinf(real-floating x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *isinf*( ) macro shall determine whether its argument value is an infinity (positive or negative). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isinf*( ) macro shall return a non-zero value if and only if its argument has an infinite value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isnan*( ), *isnormal*( ), *signbit*( ), the Base Definitions volume of IEEE Std 1003.1-2001 **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

-`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21045 **NAME**

21046 isless — test if x is less than y

21047 **SYNOPSIS**

21048 ```
#include <math.h>
```

21049 ```
int isless(real-floating x, real-floating y);
```

21050 **DESCRIPTION**

21051 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21052 conflict between the requirements described here and the ISO C standard is unintentional. This
21053 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21054 The *isless*( ) macro shall determine whether its first argument is less than its second argument.
21055 The value of *isless*(*x*, *y*) shall be equal to (*x*) < (*y*); however, unlike (*x*) < (*y*), *isless*(*x*, *y*) shall not
21056 raise the invalid floating-point exception when *x* and *y* are unordered.

21057 **RETURN VALUE**

21058 Upon successful completion, the *isless*( ) macro shall return the value of (*x*) < (*y*).

21059 If *x* or *y* is NaN, 0 shall be returned.

21060 **ERRORS**

21061 No errors are defined.

21062 **EXAMPLES**

21063 None.

21064 **APPLICATION USAGE**

21065 The relational and equality operators support the usual mathematical relationships between
21066 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21067 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21068 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21069 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21070 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21071 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21072 indicates that the argument shall be an expression of **real-floating** type.

21073 **RATIONALE**

21074 None.

21075 **FUTURE DIRECTIONS**

21076 None.

21077 **SEE ALSO**

21078 *isgreater*( ), *isgreaterequal*( ), *islessequal*( ), *islessgreater*( ), *isunordered*( ), the Base Definitions volume
21079 of IEEE Std 1003.1-2001, **<math.h>**

21080 **CHANGE HISTORY**

21081 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

21082 **NAME**

21083 islessequal — test if x is less than or equal to y

21084 **SYNOPSIS**

21085 `#include <math.h>`

21086 `int islessequal(real-floating x, real-floating y);`

21087 **DESCRIPTION**

21088 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21089 conflict between the requirements described here and the ISO C standard is unintentional. This
21090 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21091 The *islessequal*( ) macro shall determine whether its first argument is less than or equal to its
21092 second argument. The value of *islessequal*(*x*, *y*) shall be equal to (*x*) <= (*y*); however, unlike
21093 (*x*) <= (*y*), *islessequal*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are
21094 unordered.

21095 **RETURN VALUE**

21096 Upon successful completion, the *islessequal*( ) macro shall return the value of (*x*) <= (*y*).

21097 If *x* or *y* is NaN, 0 shall be returned.

21098 **ERRORS**

21099 No errors are defined.

21100 **EXAMPLES**

21101 None.

21102 **APPLICATION USAGE**

21103 The relational and equality operators support the usual mathematical relationships between
21104 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21105 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21106 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21107 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21108 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21109 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21110 indicates that the argument shall be an expression of **real-floating** type.

21111 **RATIONALE**

21112 None.

21113 **FUTURE DIRECTIONS**

21114 None.

21115 **SEE ALSO**

21116 *isgreater*( ), *isgreaterequal*( ), *isless*( ), *islessgreater*( ), *isunordered*( ), the Base Definitions volume of
21117 IEEE Std 1003.1-2001 **<math.h>**

21118 **CHANGE HISTORY**

21119 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21120 **NAME**

21121 islessgreater — test if x is less than or greater than y

21122 **SYNOPSIS**

21123 `#include <math.h>`

21124 `int islessgreater(real-floating x, real-floating y);`

21125 **DESCRIPTION**

21126 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21127 conflict between the requirements described here and the ISO C standard is unintentional. This
21128 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21129 The *islessgreater*( ) macro shall determine whether its first argument is less than or greater than
21130 its second argument. The *islessgreater*(*x*, *y*) macro is similar to (*x*) < (*y*) | | (*x*) > (*y*); however,
21131 *islessgreater*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered
21132 (nor shall it evaluate *x* and *y* twice).

21133 **RETURN VALUE**

21134 Upon successful completion, the *islessgreater*( ) macro shall return the value of
21135 (*x*) < (*y*) | | (*x*) > (*y*).

21136 If *x* or *y* is NaN, 0 shall be returned.

21137 **ERRORS**

21138 No errors are defined.

21139 **EXAMPLES**

21140 None.

21141 **APPLICATION USAGE**

21142 The relational and equality operators support the usual mathematical relationships between
21143 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21144 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21145 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21146 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21147 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21148 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21149 indicates that the argument shall be an expression of **real-floating** type.

21150 **RATIONALE**

21151 None.

21152 **FUTURE DIRECTIONS**

21153 None.

21154 **SEE ALSO**

21155 *isgreater*( ), *isgreaterequal*( ), *isless*( ), *islessequal*( ), *isunordered*( ), the Base Definitions volume of
21156 IEEE Std 1003.1-2001 **<math.h>**

21157 **CHANGE HISTORY**

21158 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

islower — test for a lowercase letter

SYNOPSIS

```
#include <ctype.h>

int islower(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *islower*( ) function shall test whether *c* is a character of class **lower** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *islower*( ) function shall return non-zero if *c* is a lowercase letter; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

Testing for a Lowercase Letter

The following example tests whether the value is a lowercase letter, based on the locale of the user, then uses it as part of a key value.

```
#include <ctype.h>
#include <stdlib.h>
#include <locale.h>
...
char *keystr;
int elementlen, len;
char c;
...
setlocale(LC_ALL, "");
...
len = 0;
while (len < elementlen) {
    c = (char) (rand() % 256);
...
    if (islower(c))
        keystr[len++] = c;
    }
...
```

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21202 **RATIONALE**

21203 None.

21204 **FUTURE DIRECTIONS**

21205 None.

21206 **SEE ALSO**

21207 *isalnum*( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ),
21208 *isxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
21209 **<ctype.h>**

21210 **CHANGE HISTORY**

21211 First released in Issue 1. Derived from Issue 1 of the SVID.

21212 **Issue 6**

21213 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

21214 An example is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

21215 **NAME**

21216 isnan — test for a NaN

21217 **SYNOPSIS**

21218
```
#include <math.h>
```

21219
```
int isnan(real-floating x);
```

21220 **DESCRIPTION**

21221 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21222 conflict between the requirements described here and the ISO C standard is unintentional. This
21223 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21224 The *isnan*( ) macro shall determine whether its argument value is a NaN. First, an argument
21225 represented in a format wider than its semantic type is converted to its semantic type. Then
21226 determination is based on the type of the argument.

21227 **RETURN VALUE**

21228 The *isnan*( ) macro shall return a non-zero value if and only if its argument has a NaN value.

21229 **ERRORS**

21230 No errors are defined.

21231 **EXAMPLES**

21232 None.

21233 **APPLICATION USAGE**

21234 None.

21235 **RATIONALE**

21236 None.

21237 **FUTURE DIRECTIONS**

21238 None.

21239 **SEE ALSO**

21240 *fpclassify*( ), *isfinite*( ), *isinf*( ), *isnormal*( ), *signbit*( ), the Base Definitions volume of
21241 IEEE Std 1003.1-2001, **<math.h>**

21242 **CHANGE HISTORY**

21243 First released in Issue 3.

21244 **Issue 5**

21245 The DESCRIPTION is updated to indicate the return value when NaN is not supported. This
21246 text was previously published in the APPLICATION USAGE section.

21247 **Issue 6**

21248 Re-written for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

21249 **NAME**

21250 isnormal — test for a normal value

21251 **SYNOPSIS**

21252
```
#include <math.h>
```

21253
```
int isnormal(real-floating x);
```

21254 **DESCRIPTION**

21255 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21256 conflict between the requirements described here and the ISO C standard is unintentional. This
21257 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21258 The *isnormal*( ) macro shall determine whether its argument value is normal (neither zero,
21259 subnormal, infinite, nor NaN). First, an argument represented in a format wider than its
21260 semantic type is converted to its semantic type. Then determination is based on the type of the
21261 argument.

21262 **RETURN VALUE**

21263 The *isnormal*( ) macro shall return a non-zero value if and only if its argument has a normal
21264 value.

21265 **ERRORS**

21266 No errors are defined.

21267 **EXAMPLES**

21268 None.

21269 **APPLICATION USAGE**

21270 None.

21271 **RATIONALE**

21272 None.

21273 **FUTURE DIRECTIONS**

21274 None.

21275 **SEE ALSO**

21276 *fpclassify*( ), *isfinite*( ), *isinf*( ), *isnan*( ), *signbit*( ), the Base Definitions volume of
21277 IEEE Std 1003.1-2001, **<math.h>**

21278 **CHANGE HISTORY**

21279 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

isprint — test for a printable character

SYNOPSIS

```
#include <ctype.h>

int isprint(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *isprint*( ) function shall test whether *c* is a character of class **print** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isprint*( ) function shall return non-zero if *c* is a printable character; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

ispunct — test for a punctuation character

SYNOPSIS

```
#include <ctype.h>

int ispunct(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *ispunct*( ) function shall test whether *c* is a character of class **punct** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *ispunct*( ) function shall return non-zero if *c* is a punctuation character; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

isspace — test for a white-space character

SYNOPSIS

```
#include <ctype.h>

int isspace(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *isspace*( ) function shall test whether *c* is a character of class **space** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *isspace*( ) function shall return non-zero if *c* is a white-space character; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21387 **NAME**

21388 isunordered — test if arguments are unordered

21389 **SYNOPSIS**

21390
```
#include <math.h>
```

21391
```
int isunordered(real-floating x, real-floating y);
```

21392 **DESCRIPTION**

21393 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21394 conflict between the requirements described here and the ISO C standard is unintentional. This
21395 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21396 The *isunordered*( ) macro shall determine whether its arguments are unordered.

21397 **RETURN VALUE**

21398 Upon successful completion, the *isunordered*( ) macro shall return 1 if its arguments are
21399 unordered, and 0 otherwise.

21400 If *x* or *y* is NaN, 0 shall be returned.

21401 **ERRORS**

21402 No errors are defined.

21403 **EXAMPLES**

21404 None.

21405 **APPLICATION USAGE**

21406 The relational and equality operators support the usual mathematical relationships between
21407 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21408 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21409 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21410 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21411 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21412 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21413 indicates that the argument shall be an expression of **real-floating** type.

21414 **RATIONALE**

21415 None.

21416 **FUTURE DIRECTIONS**

21417 None.

21418 **SEE ALSO**

21419 *isgreater*( ), *isgreaterequal*( ), *isless*( ), *islessequal*( ), *islessgreater*( ), the Base Definitions volume of
21420 IEEE Std 1003.1-2001, **<math.h>**

21421 **CHANGE HISTORY**

21422 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

21423 **NAME**

21424 isupper — test for an uppercase letter

21425 **SYNOPSIS**

21426
```
#include <ctype.h>
```

21427
```
int isupper(int c);
```

21428 **DESCRIPTION**

21429 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21430 conflict between the requirements described here and the ISO C standard is unintentional. This
21431 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21432 The *isupper*( ) function shall test whether *c* is a character of class **upper** in the program's current
21433 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21434 The *c* argument is an **int**, the value of which the application shall ensure is a character
21435 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
21436 any other value, the behavior is undefined.

21437 **RETURN VALUE**

21438 The *isupper*( ) function shall return non-zero if *c* is an uppercase letter; otherwise, it shall return 0.

21439 **ERRORS**

21440 No errors are defined.

21441 **EXAMPLES**

21442 None.

21443 **APPLICATION USAGE**

21444 To ensure applications portability, especially across natural languages, only this function and
21445 those listed in the SEE ALSO section should be used for character classification.

21446 **RATIONALE**

21447 None.

21448 **FUTURE DIRECTIONS**

21449 None.

21450 **SEE ALSO**

21451 *isalnum*( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isxdigit*( ),
21452 *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

21453 **CHANGE HISTORY**

21454 First released in Issue 1. Derived from Issue 1 of the SVID.

21455 **Issue 6**

21456 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21457 **NAME**

21458 iswalnum — test for an alphanumeric wide-character code

21459 **SYNOPSIS**

21460
```
#include <wctype.h>
```

21461
```
int iswalnum(wint_t wc);
```

21462 **DESCRIPTION**

21463 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21464 conflict between the requirements described here and the ISO C standard is unintentional. This
21465 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21466 The *iswalnum*( ) function shall test whether *wc* is a wide-character code representing a character
21467 of class **alpha** or **digit** in the program's current locale; see the Base Definitions volume of
21468 IEEE Std 1003.1-2001, Chapter 7, Locale.

21469 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21470 code corresponding to a valid character in the current locale, or equal to the value of the macro
21471 WEOF. If the argument has any other value, the behavior is undefined.

21472 **RETURN VALUE**

21473 The *iswalnum*( ) function shall return non-zero if *wc* is an alphanumeric wide-character code;
21474 otherwise, it shall return 0.

21475 **ERRORS**

21476 No errors are defined.

21477 **EXAMPLES**

21478 None.

21479 **APPLICATION USAGE**

21480 To ensure applications portability, especially across natural languages, only this function and
21481 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21482 **RATIONALE**

21483 None.

21484 **FUTURE DIRECTIONS**

21485 None.

21486 **SEE ALSO**

21487 *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ),
21488 *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21489 IEEE Std 1003.1-2001, Chapter 7, Locale, **<stdio.h>**, **<wchar.h>**, **<wctype.h>**

21490 **CHANGE HISTORY**

21491 First released as a World-wide Portability Interface in Issue 4.

21492 **Issue 5**

21493 The following change has been made in this issue for alignment with
21494 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21495 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21496 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21497 **Issue 6**

21498 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

21499 **NAME**

21500 iswalpha — test for an alphabetic wide-character code

21501 **SYNOPSIS**

21502
```
#include <wctype.h>
```

21503
```
int iswalpha(wint_t wc);
```

21504 **DESCRIPTION**

21505 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21506 conflict between the requirements described here and the ISO C standard is unintentional. This
21507 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21508 The *iswalpha*( ) function shall test whether *wc* is a wide-character code representing a character of
21509 class **alpha** in the program's current locale; see the Base Definitions volume of
21510 IEEE Std 1003.1-2001, Chapter 7, Locale.

21511 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21512 code corresponding to a valid character in the current locale, or equal to the value of the macro
21513 WEOF. If the argument has any other value, the behavior is undefined.

21514 **RETURN VALUE**

21515 The *iswalpha*( ) function shall return non-zero if *wc* is an alphabetic wide-character code;
21516 otherwise, it shall return 0.

21517 **ERRORS**

21518 No errors are defined.

21519 **EXAMPLES**

21520 None.

21521 **APPLICATION USAGE**

21522 To ensure applications portability, especially across natural languages, only this function and
21523 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21524 **RATIONALE**

21525 None.

21526 **FUTURE DIRECTIONS**

21527 None.

21528 **SEE ALSO**

21529 *iswalnum*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ),
21530 *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21531 IEEE Std 1003.1-2001, Chapter 7, Locale, **<stdio.h>**, **<wchar.h>**, **<wctype.h>**

21532 **CHANGE HISTORY**

21533 First released in Issue 4.

21534 **Issue 5**

21535 The following change has been made in this issue for alignment with
21536 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21537 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21538 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21539 **Issue 6**

21540 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21541 **NAME**

21542 iswblank — test for a blank wide-character code

21543 **SYNOPSIS**

21544
```
#include <wctype.h>
```

21545
```
int iswblank(wint_t wc);
```

21546 **DESCRIPTION**

21547 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21548 conflict between the requirements described here and the ISO C standard is unintentional. This
21549 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21550 The *iswblank*( ) function shall test whether *wc* is a wide-character code representing a character of
21551 class **blank** in the program's current locale; see the Base Definitions volume of
21552 IEEE Std 1003.1-2001, Chapter 7, Locale.

21553 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21554 code corresponding to a valid character in the current locale, or equal to the value of the macro
21555 WEOF. If the argument has any other value, the behavior is undefined.

21556 **RETURN VALUE**

21557 The *iswblank*( ) function shall return non-zero if *wc* is a blank wide-character code; otherwise, it
21558 shall return 0.

21559 **ERRORS**

21560 No errors are defined.

21561 **EXAMPLES**

21562 None.

21563 **APPLICATION USAGE**

21564 To ensure applications portability, especially across natural languages, only this function and
21565 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21566 **RATIONALE**

21567 None.

21568 **FUTURE DIRECTIONS**

21569 None.

21570 **SEE ALSO**

21571 *iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ),
21572 *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21573 IEEE Std 1003.1-2001, Chapter 7, Locale, **<stdio.h>**, **<wchar.h>**, **<wctype.h>**

21574 **CHANGE HISTORY**

21575 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

21576 **NAME**

21577 iswcntrl — test for a control wide-character code

21578 **SYNOPSIS**

21579
```
#include <wctype.h>
```

21580
```
int iswcntrl(wint_t wc);
```

21581 **DESCRIPTION**

21582 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21583 conflict between the requirements described here and the ISO C standard is unintentional. This
21584 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21585 The *iswcntrl*( ) function shall test whether *wc* is a wide-character code representing a character of
21586 class **cntrl** in the program's current locale; see the Base Definitions volume of
21587 IEEE Std 1003.1-2001, Chapter 7, Locale.

21588 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21589 code corresponding to a valid character in the current locale, or equal to the value of the macro
21590 WEOF. If the argument has any other value, the behavior is undefined.

21591 **RETURN VALUE**

21592 The *iswcntrl*( ) function shall return non-zero if *wc* is a control wide-character code; otherwise, it
21593 shall return 0.

21594 **ERRORS**

21595 No errors are defined.

21596 **EXAMPLES**

21597 None.

21598 **APPLICATION USAGE**

21599 To ensure applications portability, especially across natural languages, only this function and
21600 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21601 **RATIONALE**

21602 None.

21603 **FUTURE DIRECTIONS**

21604 None.

21605 **SEE ALSO**

21606 *iswalnum*( ), *iswalpha*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ),
21607 *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21608 IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

21609 **CHANGE HISTORY**

21610 First released in Issue 4.

21611 **Issue 5**

21612 The following change has been made in this issue for alignment with
21613 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21614 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21615 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21616 **Issue 6**

21617 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

iswctype — test character for a specified class

SYNOPSIS

```
#include <wctype.h>

int iswctype(wint_t wc, wctype_t charclass);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *iswctype*( ) function shall determine whether the wide-character code *wc* has the character class *charclass*, returning true or false. The *iswctype*( ) function is defined on WEOF and wide-character codes corresponding to the valid character encodings in the current locale. If the *wc* argument is not in the domain of the function, the result is undefined. If the value of *charclass* is invalid (that is, not obtained by a call to *wctype*( ) or *charclass* is invalidated by a subsequent call to *setlocale*( ) that has affected category *LC_CTYPE*) the result is unspecified.

RETURN VALUE

The *iswctype*( ) function shall return non-zero (true) if and only if *wc* has the property described by *charclass*. CX If *charclass* is 0, *iswctype*( ) shall return 0.

ERRORS

No errors are defined.

EXAMPLES

Testing for a Valid Character

```
#include <wctype.h>
...
int yes_or_no;
wint_t wc;
wctype_t valid_class;
...
if ((valid_class=wctype("vowel")) == (wctype_t)0)
    /* Invalid character class. */
yes_or_no=iswctype(wc,valid_class);
```

APPLICATION USAGE

The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper", and "xdigit" are reserved for the standard character classes. In the table below, the functions in the left column are equivalent to the functions in the right column.

```
iswalnum(wc)    iswctype(wc, wctype("alnum"))
iswalpha(wc)    iswctype(wc, wctype("alpha"))
iswblank(wc)    iswctype(wc, wctype("blank"))
iswcntrl(wc)    iswctype(wc, wctype("cntrl"))
iswdigit(wc)    iswctype(wc, wctype("digit"))
iswgraph(wc)    iswctype(wc, wctype("graph"))
iswlower(wc)    iswctype(wc, wctype("lower"))
iswprint(wc)    iswctype(wc, wctype("print"))
iswpunct(wc)    iswctype(wc, wctype("punct"))
iswspace(wc)    iswctype(wc, wctype("space"))
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
iswupper(wc)    iswctype(wc, wctype("upper"))
iswxdigit(wc)   iswctype(wc, wctype("xdigit"))
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *wctype*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released as World-wide Portability Interfaces in Issue 4.

Issue 5

The following change has been made in this issue for alignment with ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The behavior of *n*=0 is now described.

An example is added.

A new function, *iswblank*( ), is added to the list in the APPLICATION USAGE.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

21685 **NAME**

21686 iswdigit — test for a decimal digit wide-character code

21687 **SYNOPSIS**

21688
```
#include <wctype.h>
```

21689
```
int iswdigit(wint_t wc);
```

21690 **DESCRIPTION**

21691 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21692 conflict between the requirements described here and the ISO C standard is unintentional. This
21693 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21694 The *iswdigit*( ) function shall test whether *wc* is a wide-character code representing a character of
21695 class **digit** in the program's current locale; see the Base Definitions volume of
21696 IEEE Std 1003.1-2001, Chapter 7, Locale.

21697 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21698 code corresponding to a valid character in the current locale, or equal to the value of the macro
21699 WEOF. If the argument has any other value, the behavior is undefined.

21700 **RETURN VALUE**

21701 The *iswdigit*( ) function shall return non-zero if *wc* is a decimal digit wide-character code;
21702 otherwise, it shall return 0.

21703 **ERRORS**

21704 No errors are defined.

21705 **EXAMPLES**

21706 None.

21707 **APPLICATION USAGE**

21708 To ensure applications portability, especially across natural languages, only this function and
21709 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21710 **RATIONALE**

21711 None.

21712 **FUTURE DIRECTIONS**

21713 None.

21714 **SEE ALSO**

21715 *iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ),
21716 *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21717 IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

21718 **CHANGE HISTORY**

21719 First released in Issue 4.

21720 **Issue 5**

21721 The following change has been made in this issue for alignment with
21722 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21723 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21724 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21725 **Issue 6**

21726 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

21727 **NAME**

21728 iswgraph — test for a visible wide-character code

21729 **SYNOPSIS**

21730
```
#include <wctype.h>
```

21731
```
int iswgraph(wint_t wc);
```

21732 **DESCRIPTION**

21733 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21734 conflict between the requirements described here and the ISO C standard is unintentional. This
21735 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21736 The *iswgraph*( ) function shall test whether *wc* is a wide-character code representing a character
21737 of class **graph** in the program's current locale; see the Base Definitions volume of
21738 IEEE Std 1003.1-2001, Chapter 7, Locale.

21739 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21740 code corresponding to a valid character in the current locale, or equal to the value of the macro
21741 WEOF. If the argument has any other value, the behavior is undefined.

21742 **RETURN VALUE**

21743 The *iswgraph*( ) function shall return non-zero if *wc* is a wide-character code with a visible
21744 representation; otherwise, it shall return 0.

21745 **ERRORS**

21746 No errors are defined.

21747 **EXAMPLES**

21748 None.

21749 **APPLICATION USAGE**

21750 To ensure applications portability, especially across natural languages, only this function and
21751 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21752 **RATIONALE**

21753 None.

21754 **FUTURE DIRECTIONS**

21755 None.

21756 **SEE ALSO**

21757 *iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ),
21758 *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21759 IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

21760 **CHANGE HISTORY**

21761 First released in Issue 4.

21762 **Issue 5**

21763 The following change has been made in this issue for alignment with
21764 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21765 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21766 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21767 **Issue 6**

21768 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21769 **NAME**

21770 iswlower — test for a lowercase letter wide-character code

21771 **SYNOPSIS**

21772
```
#include <wctype.h>
```

21773
```
int iswlower(wint_t wc);
```

21774 **DESCRIPTION**

21775 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21776 conflict between the requirements described here and the ISO C standard is unintentional. This
21777 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21778 The *iswlower*( ) function shall test whether *wc* is a wide-character code representing a character
21779 of class **lower** in the program's current locale; see the Base Definitions volume of
21780 IEEE Std 1003.1-2001, Chapter 7, Locale.

21781 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21782 code corresponding to a valid character in the current locale, or equal to the value of the macro
21783 WEOF. If the argument has any other value, the behavior is undefined.

21784 **RETURN VALUE**

21785 The *iswlower*( ) function shall return non-zero if *wc* is a lowercase letter wide-character code;
21786 otherwise, it shall return 0.

21787 **ERRORS**

21788 No errors are defined.

21789 **EXAMPLES**

21790 None.

21791 **APPLICATION USAGE**

21792 To ensure applications portability, especially across natural languages, only this function and
21793 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21794 **RATIONALE**

21795 None.

21796 **FUTURE DIRECTIONS**

21797 None.

21798 **SEE ALSO**

21799 *iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswprint*( ), *iswpunct*( ),
21800 *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21801 IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

21802 **CHANGE HISTORY**

21803 First released in Issue 4.

21804 **Issue 5**

21805 The following change has been made in this issue for alignment with
21806 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21807 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21808 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21809 **Issue 6**

21810 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

iswprint — test for a printable wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswprint(wint_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *iswprint*( ) function shall test whether *wc* is a wide-character code representing a character of class **print** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswprint*( ) function shall return non-zero if *wc* is a printable wide-character code; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this issue for alignment with ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21853 **NAME**

21854 iswpunct — test for a punctuation wide-character code

21855 **SYNOPSIS**

21856
```
#include <wctype.h>
```

21857
```
int iswpunct(wint_t wc);
```

21858 **DESCRIPTION**

21859 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21860 conflict between the requirements described here and the ISO C standard is unintentional. This
21861 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21862 The *iswpunct*( ) function shall test whether *wc* is a wide-character code representing a character
21863 of class **punct** in the program's current locale; see the Base Definitions volume of
21864 IEEE Std 1003.1-2001, Chapter 7, Locale.

21865 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21866 code corresponding to a valid character in the current locale, or equal to the value of the macro
21867 WEOF. If the argument has any other value, the behavior is undefined.

21868 **RETURN VALUE**

21869 The *iswpunct*( ) function shall return non-zero if *wc* is a punctuation wide-character code;
21870 otherwise, it shall return 0.

21871 **ERRORS**

21872 No errors are defined.

21873 **EXAMPLES**

21874 None.

21875 **APPLICATION USAGE**

21876 To ensure applications portability, especially across natural languages, only this function and
21877 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21878 **RATIONALE**

21879 None.

21880 **FUTURE DIRECTIONS**

21881 None.

21882 **SEE ALSO**

21883 *iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ),
21884 *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21885 IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

21886 **CHANGE HISTORY**

21887 First released in Issue 4.

21888 **Issue 5**

21889 The following change has been made in this issue for alignment with
21890 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21891 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21892 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21893 **Issue 6**

21894 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

iswspace — test for a white-space wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswspace(wint_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *iswspace*( ) function shall test whether *wc* is a wide-character code representing a character of class **space** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswspace*( ) function shall return non-zero if *wc* is a white-space wide-character code; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this issue for alignment with ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

21937 **NAME**

21938 iswupper — test for an uppercase letter wide-character code

21939 **SYNOPSIS**

21940
```
#include <wctype.h>
```

21941
```
int iswupper(wint_t wc);
```

21942 **DESCRIPTION**

21943 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21944 conflict between the requirements described here and the ISO C standard is unintentional. This
21945 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21946 The *iswupper*( ) function shall test whether *wc* is a wide-character code representing a character
21947 of class **upper** in the program's current locale; see the Base Definitions volume of
21948 IEEE Std 1003.1-2001, Chapter 7, Locale.

21949 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21950 code corresponding to a valid character in the current locale, or equal to the value of the macro
21951 WEOF. If the argument has any other value, the behavior is undefined.

21952 **RETURN VALUE**

21953 The *iswupper*( ) function shall return non-zero if *wc* is an uppercase letter wide-character code;
21954 otherwise, it shall return 0.

21955 **ERRORS**

21956 No errors are defined.

21957 **EXAMPLES**

21958 None.

21959 **APPLICATION USAGE**

21960 To ensure applications portability, especially across natural languages, only this function and
21961 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21962 **RATIONALE**

21963 None.

21964 **FUTURE DIRECTIONS**

21965 None.

21966 **SEE ALSO**

21967 *iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ),
21968 *iswpunct*( ), *iswspace*( ), *iswxdigit*( ), *setlocale*( ), the Base Definitions volume of
21969 IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

21970 **CHANGE HISTORY**

21971 First released in Issue 4.

21972 **Issue 5**

21973 The following change has been made in this issue for alignment with
21974 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

21975 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21976 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21977 **Issue 6**

21978 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

iswxdigit — test for a hexadecimal digit wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswxdigit(wint_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *iswxdigit*( ) function shall test whether *wc* is a wide-character code representing a character of class **xdigit** in the program's current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the current locale, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

RETURN VALUE

The *iswxdigit*( ) function shall return non-zero if *wc* is a hexadecimal digit wide-character code; otherwise, it shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<wchar.h>**, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this issue for alignment with ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22021 **NAME**

22022 isxdigit — test for a hexadecimal digit

22023 **SYNOPSIS**

22024
```
#include <ctype.h>
```

22025
```
int isxdigit(int c);
```

22026 **DESCRIPTION**

22027 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22028 conflict between the requirements described here and the ISO C standard is unintentional. This
22029 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22030 The *isxdigit*( ) function shall test whether *c* is a character of class **xdigit** in the program's current
22031 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

22032 The *c* argument is an **int**, the value of which the application shall ensure is a character
22033 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
22034 any other value, the behavior is undefined.

22035 **RETURN VALUE**

22036 The *isxdigit*( ) function shall return non-zero if *c* is a hexadecimal digit; otherwise, it shall return
22037 0.

22038 **ERRORS**

22039 No errors are defined.

22040 **EXAMPLES**

22041 None.

22042 **APPLICATION USAGE**

22043 To ensure applications portability, especially across natural languages, only this function and
22044 those listed in the SEE ALSO section should be used for character classification.

22045 **RATIONALE**

22046 None.

22047 **FUTURE DIRECTIONS**

22048 None.

22049 **SEE ALSO**

22050 *isalnum*( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ),
22051 the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

22052 **CHANGE HISTORY**

22053 First released in Issue 1. Derived from Issue 1 of the SVID.

22054 **Issue 6**

22055 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

22056 **NAME**

22057 j0, j1, jn — Bessel functions of the first kind

22058 **SYNOPSIS**

22059 XSI
```
#include <math.h>

double j0(double x);
double j1(double x);
double jn(int n, double x);
```
22063

22064 **DESCRIPTION**

22065 The *j0*( ), *j1*( ), and *jn*( ) functions shall compute Bessel functions of *x* of the first kind of orders 0,
22066 1, and *n*, respectively.

22067 An application wishing to check for error situations should set *errno* to zero and call
22068 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
22069 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
22070 zero, an error has occurred.

22071 **RETURN VALUE**

22072 Upon successful completion, these functions shall return the relevant Bessel value of *x* of the
22073 first kind.

22074 If the *x* argument is too large in magnitude, or the correct result would cause underflow, 0 shall
22075 be returned and a range error may occur.

22076 If *x* is NaN, a NaN shall be returned.

22077 **ERRORS**

22078 These functions may fail if:

22079 Range Error — The value of *x* was too large in magnitude, or an underflow occurred.

22080 22081 22082 22083 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

22084 No other errors shall occur.

22085 **EXAMPLES**

22086 None.

22087 **APPLICATION USAGE**

22088 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
22089 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22090 **RATIONALE**

22091 None.

22092 **FUTURE DIRECTIONS**

22093 None.

22094 **SEE ALSO**

22095 *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *y0*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
22096 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22097 **CHANGE HISTORY**

22098 First released in Issue 1. Derived from Issue 1 of the SVID.

22099 **Issue 5**

22100 The DESCRIPTION is updated to indicate how an application should check for an error. This
22101 text was previously published in the APPLICATION USAGE section.

22102 **Issue 6**

22103 The may fail [EDOM] error is removed for the case for NaN.

22104 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
22105 with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

22106 **NAME**

22107 jrand48 — generate a uniformly distributed pseudo-random long signed integer

22108 **SYNOPSIS**

22109 XSI
```
#include <stdlib.h>

long jrand48(unsigned short xsubi[3]);
```
22110

22111

22112 **DESCRIPTION**

22113 Refer to *drand48*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

kill — send a signal to a process or a group of processes

SYNOPSIS

CX
```
#include <signal.h>

int kill(pid_t pid, int sig);
```

DESCRIPTION

The *kill*( ) function shall send a signal to a process or a group of processes specified by *pid*. The signal to be sent is specified by *sig* and is either one from the list given in **<signal.h>** or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of *pid*.

For a process to have permission to send a signal to a process designated by *pid*, unless the sending process has appropriate privileges, the real or effective user ID of the sending process shall match the real or saved set-user-ID of the receiving process.

If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal.

If *pid* is –1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for which the process has permission to send that signal.

If *pid* is negative, but not –1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the absolute value of *pid*, and for which the process has permission to send a signal.

If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait*( ) function for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending thread before *kill*( ) returns.

The user ID tests described above shall not be applied when sending SIGCONT to a process that is a member of the same session as the sending process.

An implementation that provides extended security controls may impose further implementation-defined restrictions on the sending of signals, including the null signal. In particular, the system may deny the existence of some or all of the processes specified by *pid*.

The *kill*( ) function is successful if the process has permission to send *sig* to any of the processes specified by *pid*. If *kill*( ) fails, no signal shall be sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *kill*( ) function shall fail if:

[EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.

[EPERM] The process does not have permission to send the signal to any receiving process.

[ESRCH] No process or process group can be found corresponding to that specified by *pid*.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The semantics for permission checking for *kill*( ) differed between System V and most other implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of IEEE Std 1003.1-2001 agree with System V. Specifically, a set-user-ID process cannot protect itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice allows the user who starts an application to send it signals even if it changes its effective user ID. The other semantics give more power to an application that wants to protect itself from the user who ran it.

Some implementations provide semantic extensions to the *kill*( ) function when the absolute value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a flag to *kill*( ). Since most implementations return [ESRCH] in this case, this behavior is not included in this volume of IEEE Std 1003.1-2001, although a conforming implementation could provide such an extension.

The implementation-defined processes to which a signal cannot be sent may include the scheduler or *init*.

There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the calling process and that signal is not blocked, that signal would be delivered before *kill*( ) returns. This would permit a process to call *kill*( ) and be guaranteed that the call never return. However, historical implementations that provide only the *signal*( ) function make only the weaker guarantee in this volume of IEEE Std 1003.1-2001, because they only deliver one signal each time a process enters the kernel. Modifications to such implementations to support the *sigaction*( ) function generally require entry to the kernel following return from a signal-catching function, in order to restore the signal mask. Such modifications have the effect of satisfying the stronger requirement, at least when *sigaction*( ) is used, but not necessarily when *signal*( ) is used. The developers of this volume of IEEE Std 1003.1-2001 considered making the stronger requirement except when *signal*( ) is used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the stronger requirement whenever possible. In practice, the weaker requirement is the same, except in the rare case when two signals arrive during a very short window. This reasoning also applies to a similar requirement for *sigprocmask*( ).

In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID security checks. This allows a job control shell to continue a job even if processes in the job have altered their user IDs (as in the *su* command). In keeping with the addition of the concept of sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any process in the same session regardless of user ID security checks. This is less restrictive than BSD in the sense that ancestor processes (in the same session) can now be the recipient. It is more restrictive than BSD in the sense that descendant processes that form new sessions are now subject to the user ID checks. A similar relaxation of security is not necessary for the other job control signals since those signals are typically sent by the terminal driver in recognition of special characters being typed; the terminal driver bypasses all security checks.

In secure implementations, a process may be restricted from sending a signal to a process having a different security label. In order to prevent the existence or nonexistence of a process from being used as a covert channel, such processes should appear nonexistent to the sender; that is, [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Existing implementations vary on the result of a *kill*( ) with *pid* indicating an inactive process (a terminated process that has not been waited for by its parent). Some indicate success on such a call (subject to permission checking), while others give an error of [ESRCH]. Since the definition of process lifetime in this volume of IEEE Std 1003.1-2001 covers inactive processes, the [ESRCH] error as described is inappropriate in this case. In particular, this means that an application cannot have a parent process check for termination of a particular child with *kill*( ). (Usually this is done with the null signal; this can be done reliably with *waitpid*( ).)

There is some belief that the name *kill*( ) is misleading, since the function is not always intended to cause process termination. However, the name is common to all historical implementations, and any change would be in conflict with the goal of minimal changes to existing application code.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid( ), *raise*( ), *setsid*( ), *sigaction*( ), *sigqueue*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-ID of the calling process is checked in place of its effective user ID. This is a FIPS requirement.
- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The behavior when *pid* is –1 is now specified. It was previously explicitly unspecified in the POSIX.1-1988 standard.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

22238 **NAME**

22239 killpg — send a signal to a process group

22240 **SYNOPSIS**

22241 XSI
```
#include <signal.h>

int killpg(pid_t pgrp, int sig);
```
22242

22243

22244 **DESCRIPTION**

22245 The *killpg*( ) function shall send the signal specified by *sig* to the process group specified by *pgrp*.

22246 If *pgrp* is greater than 1, *killpg*(*pgrp*, *sig*) shall be equivalent to *kill*(–*pgrp*, *sig*). If *pgrp* is less than or
22247 equal to 1, the behavior of *killpg*( ) is undefined.

22248 **RETURN VALUE**

22249 Refer to *kill*( ).

22250 **ERRORS**

22251 Refer to *kill*( ).

22252 **EXAMPLES**

22253 None.

22254 **APPLICATION USAGE**

22255 None.

22256 **RATIONALE**

22257 None.

22258 **FUTURE DIRECTIONS**

22259 None.

22260 **SEE ALSO**

22261 *getpgid*( ), *getpid*( ), *kill*( ), *raise*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

22262 **CHANGE HISTORY**

22263 First released in Issue 4, Version 2.

22264 **Issue 5**

22265 Moved from X/OPEN UNIX extension to BASE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22266 **NAME**

22267 l64a — convert a 32-bit integer to a radix-64 ASCII string

22268 **SYNOPSIS**

22269 XSI
```
#include <stdlib.h>
```

22270
```
char *l64a(long value);
```

22271

22272 **DESCRIPTION**

22273 Refer to *a64l*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

22274 **NAME**

22275 labs, llabs — return a long integer absolute value

22276 **SYNOPSIS**

22277 `#include <stdlib.h>`

22278 `long labs(long i);`
22279 `long long llabs(long long i);`

22280 **DESCRIPTION**

22281 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22282 conflict between the requirements described here and the ISO C standard is unintentional. This
22283 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22284 The *labs*( ) function shall compute the absolute value of the **long** integer operand *i*. The *llabs*( )
22285 function shall compute the absolute value of the **long long** integer operand *i*. If the result cannot
22286 be represented, the behavior is undefined.

22287 **RETURN VALUE**

22288 The *labs*( ) function shall return the absolute value of the **long** integer operand. The *labs*( )
22289 function shall return the absolute value of the **long long** integer operand.

22290 **ERRORS**

22291 No errors are defined.

22292 **EXAMPLES**

22293 None.

22294 **APPLICATION USAGE**

22295 None.

22296 **RATIONALE**

22297 None.

22298 **FUTURE DIRECTIONS**

22299 None.

22300 **SEE ALSO**

22301 *abs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

22302 **CHANGE HISTORY**

22303 First released in Issue 4. Derived from the ISO C standard.

22304 **Issue 6**

22305 The *llabs*( ) function is added for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22306 **NAME**

22307 lchown — change the owner and group of a symbolic link

22308 **SYNOPSIS**

22309 XSI
```
#include <unistd.h>

int lchown(const char *path, uid_t owner, gid_t group);
```
22310

22311

22312 **DESCRIPTION**

22313 The *lchown*( ) function shall be equivalent to *chown*( ), except in the case where the named file is a
22314 symbolic link. In this case, *lchown*( ) shall change the ownership of the symbolic link file itself,
22315 while *chown*( ) changes the ownership of the file or directory to which the symbolic link refers.

22316 **RETURN VALUE**

22317 Upon successful completion, *lchown*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to
22318 indicate an error.

22319 **ERRORS**

22320 The *lchown*( ) function shall fail if:

22321 [EACCES] Search permission is denied on a component of the path prefix of *path*.

22322 [EINVAL] The owner or group ID is not a value supported by the implementation.

22323 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
22324 argument.

22325 [ENAMETOOLONG]
22326 The length of a pathname exceeds {PATH_MAX} or a pathname component is
22327 longer than {NAME_MAX}.

22328 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

22329 [ENOTDIR] A component of the path prefix of *path* is not a directory.

22330 [EOPNOTSUPP] The *path* argument names a symbolic link and the implementation does not
22331 support setting the owner or group of a symbolic link.

22332 [EPERM] The effective user ID does not match the owner of the file and the process
22333 does not have appropriate privileges.

22334 [EROFS] The file resides on a read-only file system.

22335 The *lchown*( ) function may fail if:

22336 [EIO] An I/O error occurred while reading or writing to the file system.

22337 [EINTR] A signal was caught during execution of the function.

22338 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
22339 resolution of the *path* argument.

22340 [ENAMETOOLONG]
22341 Pathname resolution of a symbolic link produced an intermediate result
22342 whose length exceeds {PATH_MAX}.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Changing the Current Owner of a File

The following example shows how to change the ownership of the symbolic link named **/modules/pass1** to the user ID associated with ''jones'' and the group ID associated with ''cnd''.

The numeric value for the user ID is obtained by using the *getpwnam*( ) function. The numeric value for the group ID is obtained by using the *getgrnam*( ) function.

```
#include <sys/types.h>
#include <unistd.h>
#include <pwd.h>
#include <grp.h>

struct passwd *pwd;
struct group  *grp;
char          *path = "/modules/pass1";
...
pwd = getpwnam("jones");
grp = getgrnam("cnd");
lchown(path, pwd->pw_uid, grp->gr_gid);
```

APPLICATION USAGE

On implementations which support symbolic links as directory entries rather than files, *lchown*( ) may fail.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( ), *symlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

The Open Group Base Resolution bwg2001-013 is applied, adding wording to the APPLICATION USAGE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22378 **NAME**

22379 lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

22380 **SYNOPSIS**

22381 XSI
```
#include <stdlib.h>
```

22382
```
void lcong48(unsigned short param[7]);
```

22383

22384 **DESCRIPTION**

22385 Refer to *drand48*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ldexp, ldexpf, ldexpl — load exponent of a floating-point number

SYNOPSIS

```
#include <math.h>

double ldexp(double x, int exp);
float ldexpf(float x, int exp);
long double ldexpl(long double x, int exp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the quantity $x * 2^{exp}$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power *exp*.

If these functions would cause overflow, a range error shall occur and *ldexp*( ), *ldexpf*( ), and *ldexpl*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*), respectively.

If the correct value would cause underflow, and is not representable, a range error may occur,
MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

If *exp* is 0, *x* shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Range Error — The result overflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error — The result underflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *frexp*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *ldexpf*( ) and *ldexpl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ldiv, lldiv — compute quotient and remainder of a long division

SYNOPSIS

```
#include <stdlib.h>

ldiv_t ldiv(long numer, long denom);
lldiv_t lldiv(long long numer, long long denom);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the quotient and remainder of the division of the numerator *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long** integer (for the *ldiv*( ) function) or **long long** integer (for the *lldiv*( ) function) of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, *quot* * *denom*+*rem* shall equal *numer*.

RETURN VALUE

The *ldiv*( ) function shall return a structure of type **ldiv_t**, comprising both the quotient and the remainder. The structure shall include the following members, in any order:

```
long   quot;    /* Quotient */
long   rem;     /* Remainder */
```

The *lldiv*( ) function shall return a structure of type **lldiv_t**, comprising both the quotient and the remainder. The structure shall include the following members, in any order:

```
long long   quot;    /* Quotient */
long long   rem;     /* Remainder */
```

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

div( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

Issue 6

The *lldiv*( ) function is added for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22493 **NAME**

22494 lfind — find entry in a linear search table

22495 **SYNOPSIS**

22496 XSI
```
#include <search.h>

void *lfind(const void *key, const void *base, size_t *nelp,
    size_t width, int (*compar)(const void *, const void *));
```

22499

22500 **DESCRIPTION**

22501 Refer to *lsearch*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`-,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

22502 **NAME**

22503 lgamma, lgammaf, lgammal — log gamma function

22504 **SYNOPSIS**

22505
```
#include <math.h>
```

22506
```
double lgamma(double x);
```
22507
```
float lgammaf(float x);
```
22508
```
long double lgammal(long double x);
```
22509 XSI
```
extern int signgam;
```
22510

22511 **DESCRIPTION**

22512 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22513 conflict between the requirements described here and the ISO C standard is unintentional. This
22514 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22515 These functions shall compute $\log_e|\Gamma(x)|$ where $\Gamma(x)$ is defined as $\int_0^\infty e^{-t}t^{x-1}dt$. The argument x
22516
22517 need not be a non-positive integer ($\Gamma(x)$ is defined over the reals, except the non-positive
22518 integers).

22519 XSI The sign of $\Gamma(x)$ is returned in the external integer *signgam*.

22520 CX These functions need not be reentrant. A function that is not required to be reentrant is not
22521 required to be thread-safe.

22522 An application wishing to check for error situations should set *errno* to zero and call
22523 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
22524 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
22525 zero, an error has occurred.

22526 **RETURN VALUE**

22527 Upon successful completion, these functions shall return the logarithmic gamma of *x*.

22528 If *x* is a non-positive integer, a pole error shall occur and *lgamma*( ), *lgammaf*( ), and *lgammal*( )
22529 shall return +HUGE_VAL, +HUGE_VALF, and +HUGE_VALL, respectively.

22530 If the correct value would cause overflow, a range error shall occur and *lgamma*( ), *lgammaf*( ),
22531 and *lgammal*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (having the same
22532 sign as the correct value), respectively.

22533 MX If *x* is NaN, a NaN shall be returned.

22534 If *x* is 1 or 2, +0 shall be returned.

22535 If *x* is ±Inf, +Inf shall be returned.

22536 **ERRORS**

22537 These functions shall fail if:

22538 Pole Error — The *x* argument is a negative integer or zero.

22539 22540 22541 22542 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

22543 Range Error — The result overflows.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *lgamma*( ) function is no longer marked as an extension.

The *lgammaf*( ) and *lgammal*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

XSI extensions are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

link — link to a file

SYNOPSIS

```
#include <unistd.h>

int link(const char *path1, const char *path2);
```

DESCRIPTION

The *link*( ) function shall create a new link (directory entry) for the existing file, *path1*.

The *path1* argument points to a pathname naming an existing file. The *path2* argument points to a pathname naming the new directory entry to be created. The *link*( ) function shall atomically create a new link for the existing file and the link count of the file shall be incremented by one.

If *path1* names a directory, *link*( ) shall fail unless the process has appropriate privileges and the implementation supports using *link*( ) on directories.

Upon successful completion, *link*( ) shall mark for update the *st_ctime* field of the file. Also, the *st_ctime* and *st_mtime* fields of the directory that contains the new entry shall be marked for update.

If *link*( ) fails, no link shall be created and the link count of the file shall remain unchanged.

The implementation may require that the calling process has permission to access the existing file.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *link*( ) function shall fail if:

[EACCES] A component of either path prefix denies search permission, or the requested link requires writing in a directory that denies write permission, or the calling process does not have permission to access the existing file and this is required by the implementation.

[EEXIST] The *path2* argument resolves to an existing file or refers to a symbolic link.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or *path2* argument.

[EMLINK] The number of links to the file named by *path1* would exceed {LINK_MAX}.

[ENAMETOOLONG]
The length of the *path1* or *path2* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of either path prefix does not exist; the file named by *path1* does not exist; or *path1* or *path2* points to an empty string.

[ENOSPC] The directory to contain the link cannot be extended.

[ENOTDIR] A component of either path prefix is not a directory.

[EPERM] The file named by *path1* is a directory and either the calling process does not have appropriate privileges or the implementation prohibits using *link*( ) on directories.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22616 [EROFS] The requested link requires writing in a directory on a read-only file system.

22617 [EXDEV] The link named by *path2* and the file named by *path1* are on different file
22618 systems and the implementation does not support links between file systems.

22619 XSR [EXDEV] *path1* refers to a named STREAM.

22620 The *link*( ) function may fail if:

22621 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
22622 resolution of the *path1* or *path2* argument.

22623 [ENAMETOOLONG]
22624 As a result of encountering a symbolic link in resolution of the *path1* or *path2*
22625 argument, the length of the substituted pathname string exceeded
22626 {PATH_MAX}.

22627 **EXAMPLES**

22628 **Creating a Link to a File**

22629 The following example shows how to create a link to a file named **/home/cnd/mod1** by creating a
22630 new directory entry named **/modules/pass1**.

```
22631 #include <unistd.h>

22632 char *path1 = "/home/cnd/mod1";
22633 char *path2 = "/modules/pass1";
22634 int   status;
22635 ...
22636 status = link (path1, path2);
```

22637 **Creating a Link to a File Within a Program**

22638 In the following program example, the *link*( ) function links the **/etc/passwd** file (defined as
22639 **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the
22640 current password file. Then, after removing the current password file (defined as
22641 **PASSWDFILE**), the new password file is saved as the current password file using the *link*( )
22642 function again.

```
22643 #include <unistd.h>

22644 #define LOCKFILE "/etc/ptmp"
22645 #define PASSWDFILE "/etc/passwd"
22646 #define SAVEFILE "/etc/opasswd"
22647 ...
22648 /* Save current password file */
22649 link (PASSWDFILE, SAVEFILE);

22650 /* Remove current password file. */
22651 unlink (PASSWDFILE);

22652 /* Save new password file as current password file. */
22653 link (LOCKFILE,PASSWDFILE);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

22654 **APPLICATION USAGE**

22655 Some implementations do allow links between file systems.

22656 **RATIONALE**

22657 Linking to a directory is restricted to the superuser in most historical implementations because
22658 this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This
22659 volume of IEEE Std 1003.1-2001 continues that philosophy by prohibiting *link*( ) and *unlink*( )
22660 from doing this. Other functions could do it if the implementor designed such an extension.

22661 Some historical implementations allow linking of files on different file systems. Wording was
22662 added to explicitly allow this optional behavior.

22663 The exception for cross-file system links is intended to apply only to links that are
22664 programmatically indistinguishable from ''hard'' links.

22665 **FUTURE DIRECTIONS**

22666 None.

22667 **SEE ALSO**

22668 *symlink*( ), *unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

22669 **CHANGE HISTORY**

22670 First released in Issue 1. Derived from Issue 1 of the SVID.

22671 **Issue 6**

22672 The following new requirements on POSIX implementations derive from alignment with the
22673 Single UNIX Specification:

22674 - The [ELOOP] mandatory error condition is added.

22675 - A second [ENAMETOOLONG] is added as an optional error condition.

22676 The following changes were made to align with the IEEE P1003.1a draft standard:

22677 - An explanation is added of the action when *path2* refers to a symbolic link.

22678 - The [ELOOP] optional error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

lio_listio — list directed I/O (**REALTIME**)

SYNOPSIS

AIO
```
#include <aio.h>

int lio_listio(int mode, struct aiocb *restrict const list[restrict],
    int nent, struct sigevent *restrict sig);
```

DESCRIPTION

The *lio_listio*( ) function shall initiate a list of I/O requests with a single function call.

The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in **<aio.h>** and determines whether the function returns when the I/O operations have been completed, or as soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall wait until all I/O is complete and the *sig* argument shall be ignored.

If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous notification shall occur, according to the *sig* argument, when all the I/O operations complete. If *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous notification occurs as specified in Section 2.4.1 (on page 28) when all the requests in *list* have completed.

The I/O requests enumerated by *list* are submitted in an unspecified order.

The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements. The array may contain NULL elements, which shall be ignored.

The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in **<aio.h>**. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode* element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read*( ) with the *aiocbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal to LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write*( ) with the *aiocbp* equal to the address of the **aiocb** structure.

The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.

The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.

The *aio_nbytes* member specifies the number of bytes of data to be transferred.

The members of the **aiocb** structure further describe the I/O operation to be performed, in a manner identical to that of the corresponding **aiocb** structure when used by the *aio_read*( ) and *aio_write*( ) functions.

The *nent* argument specifies how many elements are members of the list; that is, the length of the array.

The behavior of this function is altered according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion if synchronized I/O is enabled on the file associated with *aio_fildes*.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

RETURN VALUE

If the *mode* argument has the value LIO_NOWAIT, the *lio_listio*( ) function shall return the value zero if the I/O operations are successfully queued; otherwise, the function shall return the value –1 and set *errno* to indicate the error.

If the *mode* argument has the value LIO_WAIT, the *lio_listio*( ) function shall return the value zero when all the indicated I/O has completed successfully. Otherwise, *lio_listio*( ) shall return a value of –1 and set *errno* to indicate the error.

In either case, the return value only indicates the success or failure of the *lio_listio*( ) call itself, not the status of the individual I/O requests. In some cases one or more of the I/O requests contained in the list may fail. Failure of an individual request does not prevent completion of any other individual request. To determine the outcome of each I/O request, the application shall examine the error status associated with each **aiocb** control block. The error statuses so returned are identical to those returned as the result of an *aio_read*( ) or *aio_write*( ) function.

ERRORS

The *lio_listio*( ) function shall fail if:

[EAGAIN] The resources necessary to queue all the I/O requests were not available. The application may check the error status for each **aiocb** to determine the individual request(s) that failed.

[EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit {AIO_MAX} to be exceeded.

[EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than {AIO_LISTIO_MAX}.

[EINTR] A signal was delivered while waiting for all I/O requests to complete during an LIO_WAIT operation. Note that, since each I/O operation invoked by *lio_listio*( ) may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited. Outstanding I/O requests are not canceled, and the application shall examine each list element to determine whether the request was initiated, canceled, or completed.

[EIO] One or more of the individual I/O operations failed. The application may check the error status for each **aiocb** structure to determine the individual request(s) that failed.

In addition to the errors returned by the *lio_listio*( ) function, if the *lio_listio*( ) function succeeds or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list may have been initiated. If the *lio_listio*( ) function fails with an error code other than [EAGAIN], [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation indicated by each list element can encounter errors specific to the individual read or write function being performed. In this event, the error status for each **aiocb** control block contains the associated error code. The error codes that can be set are the same as would be set by a *read*( ) or *write*( ) function, with the following additional error codes possible:

[EAGAIN] The requested I/O operation was not queued due to resource limitations.

[ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel*( ) request.

[EFBIG] The *aiocbp->aio_lio_opcode* is LIO_WRITE, the file is a regular file, *aiocbp->aio_nbytes* is greater than 0, and the *aiocbp->aio_offset* is greater than or equal to the offset maximum in the open file description associated with

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

aiocbp->aio_fildes.

[EINPROGRESS] The requested I/O is in progress.

[EOVERFLOW] The *aiocbp->aio_lio_opcode* is LIO_READ, the file is a regular file, *aiocbp->aio_nbytes* is greater than 0, and the *aiocbp->aio_offset* is before the end-of-file and is greater than or equal to the offset maximum in the open file description associated with *aiocbp->aio_fildes*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Although it may appear that there are inconsistencies in the specified circumstances for error codes, the [EIO] error condition applies when any circumstance relating to an individual operation makes that operation fail. This might be due to a badly formulated request (for example, the *aio_lio_opcode* field is invalid, and *aio_error*( ) returns [EINVAL]) or might arise from application behavior (for example, the file descriptor is closed before the operation is initiated, and *aio_error*( ) returns [EBADF]).

The limitation on the set of error codes returned when operations from the list shall have been initiated enables applications to know when operations have been started and whether *aio_error*( ) is valid for a specific operation.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_read( ), *aio_write*( ), *aio_error*( ), *aio_return*( ), *aio_cancel*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lseek*( ), *read*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The *lio_listio*( ) function is marked as part of the Asynchronous Input and Output option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*. This change is to support large files.
- The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support large files.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *lio_listio*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

22808 **NAME**

22809 listen — listen for socket connections and limit the queue of incoming connections

22810 **SYNOPSIS**

22811 `#include <sys/socket.h>`

22812 `int listen(int socket, int backlog);`

22813 **DESCRIPTION**

22814 The *listen*( ) function shall mark a connection-mode socket, specified by the *socket* argument, as
22815 accepting connections.

22816 The *backlog* argument provides a hint to the implementation which the implementation shall use
22817 to limit the number of outstanding connections in the socket's listen queue. Implementations
22818 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*
22819 argument value shall result in a larger or equal length of the listen queue. Implementations shall
22820 support values of *backlog* up to SOMAXCONN, defined in **<sys/socket.h>**.

22821 The implementation may include incomplete connections in its listen queue. The limits on the
22822 number of incomplete connections and completed connections queued may be different.

22823 The implementation may have an upper limit on the length of the listen queue—either global or
22824 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

22825 If *listen*( ) is called with a *backlog* argument value that is less than 0, the function behaves as if it
22826 had been called with a *backlog* argument value of 0.

22827 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of
22828 the listen queue may be set to an implementation-defined minimum value.

22829 The socket in use may require the process to have appropriate privileges to use the *listen*( )
22830 function.

22831 **RETURN VALUE**

22832 Upon successful completions, *listen*( ) shall return 0; otherwise, –1 shall be returned and *errno* set
22833 to indicate the error.

22834 **ERRORS**

22835 The *listen*( ) function shall fail if:

22836 [EBADF] The *socket* argument is not a valid file descriptor.

22837 [EDESTADDRREQ]
22838 The socket is not bound to a local address, and the protocol does not support
22839 listening on an unbound socket.

22840 [EINVAL] The *socket* is already connected.

22841 [ENOTSOCK] The *socket* argument does not refer to a socket.

22842 [EOPNOTSUPP] The socket protocol does not support *listen*( ).

22843 The *listen*( ) function may fail if:

22844 [EACCES] The calling process does not have the appropriate privileges.

22845 [EINVAL] The *socket* has been shut down.

22846 [ENOBUFS] Insufficient resources are available in the system to complete the call.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22847 **EXAMPLES**

22848 None.

22849 **APPLICATION USAGE**

22850 None.

22851 **RATIONALE**

22852 None.

22853 **FUTURE DIRECTIONS**

22854 None.

22855 **SEE ALSO**

22856 *accept*( ), *connect*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

22857 **CHANGE HISTORY**

22858 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

22859 The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*
22860 argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

22861 **NAME**

22862 llabs — return a long integer absolute value

22863 **SYNOPSIS**

22864 `#include <stdlib.h>`

22865 `long long llabs(long long i);`

22866 **DESCRIPTION**

22867 Refer to *labs*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22868 **NAME**

22869 lldiv — compute quotient and remainder of a long division

22870 **SYNOPSIS**

22871 `#include <stdlib.h>`

22872 `lldiv_t lldiv(long long` *`numer`*`, long long` *`denom`*`);`

22873 **DESCRIPTION**

22874 Refer to *ldiv*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

SYNOPSIS

```
#include <math.h>

long long llrint(double x);
long long llrintf(float x);
long long llrintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long long**, a domain error shall occur and an unspecified value is returned.

If the correct value is negative and too large to represent as a **long long**, a domain error shall occur and an unspecified value is returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

floating-point exception.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *lrint*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

22927 **NAME**

22928 llround, llroundf, llroundl — round to nearest integer value

22929 **SYNOPSIS**

22930
```
#include <math.h>
```

22931
```
long long llround(double x);
```
22932
```
long long llroundf(float x);
```
22933
```
long long llroundl(long double x);
```

22934 **DESCRIPTION**

22935 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22936 conflict between the requirements described here and the ISO C standard is unintentional. This
22937 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22938 These functions shall round their argument to the nearest integer value, rounding halfway cases
22939 away from zero, regardless of the current rounding direction.

22940 An application wishing to check for error situations should set *errno* to zero and call
22941 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
22942 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
22943 zero, an error has occurred.

22944 **RETURN VALUE**

22945 Upon successful completion, these functions shall return the rounded integer value.

22946 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

22947 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

22948 If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

22949 If the correct value is positive and too large to represent as a **long long**, a domain error shall
22950 occur and an unspecified value is returned.

22951 If the correct value is negative and too large to represent as a **long long**, a domain error shall
22952 occur and an unspecified value is returned.

22953 **ERRORS**

22954 These functions shall fail if:

22955 MX Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an
22956 integer.

22957 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
22958 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
22959 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
22960 shall be raised.

22961 **EXAMPLES**

22962 None.

22963 **APPLICATION USAGE**

22964 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
22965 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22966 **RATIONALE**

22967 These functions differ from the *llrint*( ) functions in that the default rounding direction for the
22968 *llround*( ) functions round halfway cases away from zero and need not raise the inexact floating-
22969 point exception for non-integer arguments that round to within the range of the return type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22970 **FUTURE DIRECTIONS**

22971 None.

22972 **SEE ALSO**

22973 *feclearexcept*( ), *fetestexcept*( ), *lround*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
22974 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

22975 **CHANGE HISTORY**

22976 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

localeconv — return locale-specific information

SYNOPSIS

```
#include <locale.h>

struct lconv *localeconv(void);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *localeconv*( ) function shall set the components of an object with the type **struct lconv** with the values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type **char *** are pointers to strings, any of which (except **decimal_point**) can point to "", to indicate that the value is not available in the current locale or is of zero length. The members with type **char** are non-negative numbers, any of which can be {CHAR_MAX} to indicate that the value is not available in the current locale.

The members include the following:

char *decimal_point
The radix character used to format non-monetary quantities.

char *thousands_sep
The character used to separate groups of digits before the decimal-point character in formatted non-monetary quantities.

char *grouping
A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted non-monetary quantities.

char *int_curr_symbol
The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in the ISO 4217:2001 standard. The fourth character (immediately preceding the null byte) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol
The local currency symbol applicable to the current locale.

char *mon_decimal_point
The radix character used to format monetary quantities.

char *mon_thousands_sep
The separator for groups of digits before the decimal-point in formatted monetary quantities.

char *mon_grouping
A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign
The string used to indicate a non-negative valued formatted monetary quantity.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

char *negative_sign
The string used to indicate a negative valued formatted monetary quantity.

char int_frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in an internationally formatted monetary quantity.

char frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

char p_cs_precedes
Set to 1 if the **currency_symbol** precedes the value for a non-negative formatted monetary quantity. Set to 0 if the symbol succeeds the value.

char p_sep_by_space
Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a non-negative formatted monetary quantity.

char n_cs_precedes
Set to 1 if the **currency_symbol** precedes the value for a negative formatted monetary quantity. Set to 0 if the symbol succeeds the value.

char n_sep_by_space
Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a negative formatted monetary quantity.

char p_sign_posn
Set to a value indicating the positioning of the **positive_sign** for a non-negative formatted monetary quantity.

char n_sign_posn
Set to a value indicating the positioning of the **negative_sign** for a negative formatted monetary quantity.

char int_p_cs_precedes
Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a non-negative internationally formatted monetary quantity.

char int_n_cs_precedes
Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a negative internationally formatted monetary quantity.

char int_p_sep_by_space
Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the value for a non-negative internationally formatted monetary quantity.

char int_n_sep_by_space
Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the value for a negative internationally formatted monetary quantity.

char int_p_sign_posn
Set to a value indicating the positioning of the **positive_sign** for a non-negative internationally formatted monetary quantity.

char int_n_sign_posn
Set to a value indicating the positioning of the **negative_sign** for a negative internationally formatted monetary quantity.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The elements of **grouping** and **mon_grouping** are interpreted according to the following:

{CHAR_MAX}	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
other	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of **p_sep_by_space**, **n_sep_by_space**, **int_p_sep_by_space**, and **int_n_sep_by_space** are interpreted according to the following:

0 No space separates the currency symbol and value.

1 If the currency symbol and sign string are adjacent, a space separates them from the value; otherwise, a space separates the currency symbol from the value.

2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a space separates the sign string from the value.

For **int_p_sep_by_space** and **int_n_sep_by_space**, the fourth character of **int_curr_symbol** is used instead of a space.

The values of **p_sign_posn**, **n_sign_posn**, **int_p_sign_posn**, and **int_n_sign_posn** are interpreted according to the following:

0 Parentheses surround the quantity and **currency_symbol** or **int_curr_symbol**.

1 The sign string precedes the quantity and **currency_symbol** or **int_curr_symbol**.

2 The sign string succeeds the quantity and **currency_symbol** or **int_curr_symbol**.

3 The sign string immediately precedes the **currency_symbol** or **int_curr_symbol**.

4 The sign string immediately succeeds the **currency_symbol** or **int_curr_symbol**.

The implementation shall behave as if no function in this volume of IEEE Std 1003.1-2001 calls *localeconv*( ).

CX The *localeconv*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *localeconv*( ) function shall return a pointer to the filled-in object. The application shall not modify the structure pointed to by the return value which may be overwritten by a subsequent call to *localeconv*( ). In addition, calls to *setlocale*( ) with the categories *LC_ALL*, *LC_MONETARY*, or *LC_NUMERIC* may overwrite the contents of the structure.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The following table illustrates the rules which may be used by four countries to format monetary quantities.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Country	Positive Format	Negative Format	International Format
Italy	L.1.230	–L.1.230	ITL.1.230
Netherlands	F 1.234,56	F –1.234,56	NLG 1.234,56
Norway	kr1.234,56	kr1.234,56–	NOK 1.234,56
Switzerland	SFrs.1,234.56	SFrs.1,234.56C	CHF 1,234.56

For these four countries, the respective values for the monetary members of the structure returned by *localeconv*( ) are:

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	"ITL."	"NLG "	"NOK "	"CHF "
currency_symbol	"L."	"F"	"kr"	"SFrs."
mon_decimal_point	""	","	","	"."
mon_thousands_sep	"."	"."	"."	","
mon_grouping	"\3"	"\3"	"\3"	"\3"
positive_sign	""	""	""	""
negative_sign	"-"	"-"	"-"	"C"
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2
int_p_cs_precedes	1	1	1	1
int_n_cs_precedes	1	1	1	1
int_p_sep_by_space	0	0	0	0
int_n_sep_by_space	0	0	0	0
int_p_sign_posn	1	1	1	1
int_n_sign_posn	1	4	4	2

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalpha( ), *isascii*( ), *nl_langinfo*( ), *printf*( ), *scanf*( ), *setlocale*( ), *strcat*( ), *strchr*( ), *strcmp*( ), *strcoll*( ), *strcpy*( ), *strftime*( ), *strlen*( ), *strpbrk*( ), *strspn*( ), *strtok*( ), *strxfrm*( ), *strtod*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<langinfo.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

The RETURN VALUE section is rewritten to avoid use of the term ‘‘must’’.

This reference page is updated for alignment with the ISO / IEC 9899: 1999 standard.

ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to **int_curr_symbol** and updating the descriptions of **p_sep_by_space** and **n_sep_by_space**. These changes are for alignment with the ISO C standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

23150 **NAME**

23151 localtime, localtime_r — convert a time value to a broken-down local time

23152 **SYNOPSIS**

23153 ```
#include <time.h>
```

23154 ```
struct tm *localtime(const time_t *timer);
```

23155 TSF ```
struct tm *localtime_r(const time_t *restrict timer,
23156   struct tm *restrict result);
```

23157

23158 **DESCRIPTION**

23159 CX For *localtime*( ): The functionality described on this reference page is aligned with the ISO C
23160 standard. Any conflict between the requirements described here and the ISO C standard is
23161 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23162 The *localtime*( ) function shall convert the time in seconds since the Epoch pointed to by *timer*
23163 into a broken-down time, expressed as a local time. The function corrects for the timezone and
23164 CX any seasonal time adjustments. Local timezone information is used as though *localtime*( ) calls
23165 *tzset*( ).

23166 The relationship between a time in seconds since the Epoch used as an argument to *localtime*( )
23167 and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in the
23168 expression given in the definition of seconds since the Epoch (see the Base Definitions volume of
23169 IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch) corrected for timezone and any
23170 seasonal time adjustments, where the names in the structure and in the expression correspond.

23171 TSF The same relationship shall apply for *localtime_r*( ).

23172 CX The *localtime*( ) function need not be reentrant. A function that is not required to be reentrant is
23173 not required to be thread-safe.

23174 The *asctime*( ), *ctime*( ), *gmtime*( ), and *localtime*( ) functions shall return values in one of two static
23175 objects: a broken-down time structure and an array of type **char**. Execution of any of the
23176 functions may overwrite the information returned in either of these objects by any of the other
23177 functions.

23178 TSF The *localtime_r*( ) function shall convert the time in seconds since the Epoch pointed to by *timer*
23179 into a broken-down time stored in the structure to which *result* points. The *localtime_r*( ) function
23180 shall also return a pointer to that same structure.

23181 Unlike *localtime*( ), the reentrant version is not required to set *tzname*.

23182 **RETURN VALUE**

23183 Upon successful completion, the *localtime*( ) function shall return a pointer to the broken-down
23184 CX time structure. If an error is detected, *localtime*( ) shall return a null pointer and set *errno* to
23185 indicate the error.

23186 TSF Upon successful completion, *localtime_r*( ) shall return a pointer to the structure pointed to by
23187 the argument *result*.

23188 **ERRORS**

23189 The *localtime*( ) function shall fail if:

23190 CX [EOVERFLOW] The result cannot be represented.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale
```

EXAMPLES

Getting the Local Date and Time

The following example uses the *time*() function to calculate the time elapsed, in seconds, since January 1, 1970 0:00 UTC (the Epoch), *localtime*() to convert that value to a broken-down time, and *asctime*() to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t result;

    result = time(NULL);
    printf("%s%ju secs since the Epoch\n",
        asctime(localtime(&result)),
            (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```

Getting the Modification Time for a File

The following example gets the modification time for a file. The *localtime*() function converts the **time_t** value of the last modification date, obtained by a previous call to *stat*(), into a **tm** structure that contains the year, month, day, and so on.

```
#include <time.h>
...
struct stat statbuf;
...
tm = localtime(&statbuf.st_mtime);
...
```

Timing an Event

The following example gets the current time, converts it to a string using *localtime*() and *asctime*(), and prints it to standard output using *fputs*(). It then prints the number of minutes to an event being timed.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
printf("The time is ");
fputs(asctime(localtime(&now)), stdout);
printf("There are still %d minutes to the event.\n",
```

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
            minutes_to_event);
        ...
```

APPLICATION USAGE

The *localtime_r*( ) function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *getdate*( ), *gmtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *localtime*( ) function need not be reentrant is added to the DESCRIPTION.

The *localtime_r*( ) function is included for alignment with the POSIX Threads Extension.

Issue 6

The *localtime_r*( ) function is marked as part of the Thread-Safe Functions option.

Extensions beyond the ISO C standard are marked.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *localtime_r*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

Examples are added.

IEEE Std 1003.1-2001 / Cor 1-2002, item m XSH / TC1 / D6 / 32 is applied, adding the [EOVERFLOW] error.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

lockf — record locking on files

SYNOPSIS

XSI
```
#include <unistd.h>

int lockf(int fildes, int function, off_t size);
```

DESCRIPTION

The *lockf*( ) function shall lock sections of a file with advisory-mode locks. Calls to *lockf*( ) from other threads which attempt to lock the locked file section shall either return an error value or block until the section becomes unlocked. All the locks for a process are removed when the process terminates. Record locking with *lockf*( ) shall be supported for regular files and may be supported for other files.

The *fildes* argument is an open file descriptor. To establish a lock with this function, the file descriptor shall be opened with write-only permission (O_WRONLY) or with read/write permission (O_RDWR).

The *function* argument is a control value which specifies the action to be taken. The permissible values for *function* are defined in **<unistd.h>** as follows:

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

F_TEST shall detect if a lock by another process is present on the specified section.

F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

F_ULOCK shall remove locks from a section of the file.

The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be locked or unlocked starts at the current offset in the file and extends forward for a positive size or backward for a negative size (the preceding bytes up to but not including the current offset). If *size* is 0, the section from the current offset through the largest possible file offset shall be locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file to be locked because locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent locked sections would occur, the sections shall be combined into a single locked section. If the request would cause the number of locks to exceed a system-imposed limit, the request shall fail.

F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available. F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the function to fail if the section is already locked by another process.

File locks shall be released on first close by the locking process of any file descriptor for the file.

F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or to the end-of-file if *size* is (**off_t**)0. When all of a locked section is not released (that is, when the beginning or end of the area to be unlocked falls within a locked section), the remaining portions of that section shall remain locked by the process. Releasing the center portion of a locked

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

23307 section shall cause the remaining locked beginning and end portions to become two separate
23308 locked sections. If the request would cause the number of locks in the system to exceed a
23309 system-imposed limit, the request shall fail.

23310 A potential for deadlock occurs if the threads of a process controlling a locked section are
23311 blocked by accessing another process' locked section. If the system detects that deadlock would
23312 occur, *lockf*( ) shall fail with an [EDEADLK] error.

23313 The interaction between *fcntl*( ) and *lockf*( ) locks is unspecified.

23314 Blocking on a section shall be interrupted by any signal.

23315 An F_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested
23316 section is the maximum value for an object of type **off_t**, when the process has an existing lock
23317 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a
23318 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an
23319 F_ULOCK request shall attempt to unlock only the requested section.

23320 Attempting to lock a section of a file that is associated with a buffered stream produces
23321 unspecified results.

23322 **RETURN VALUE**

23323 Upon successful completion, *lockf*( ) shall return 0. Otherwise, it shall return –1, set *errno* to
23324 indicate an error, and existing locks shall not be changed.

23325 **ERRORS**

23326 The *lockf*( ) function shall fail if:

23327 [EBADF] The *fildes* argument is not a valid open file descriptor; or *function* is F_LOCK
23328 or F_TLOCK and *fildes* is not a valid file descriptor open for writing.

23329 [EACCES] or [EAGAIN]
23330 The *function* argument is F_TLOCK or F_TEST and the section is already
23331 locked by another process.

23332 [EDEADLK] The *function* argument is F_LOCK and a deadlock is detected.

23333 [EINTR] A signal was caught during execution of the function.

23334 [EINVAL] The *function* argument is not one of F_LOCK, F_TLOCK, F_TEST, or
23335 F_ULOCK; or *size* plus the current file offset is less than 0.

23336 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested
23337 section cannot be represented correctly in an object of type **off_t**.

23338 The *lockf*( ) function may fail if:

23339 [EAGAIN] The *function* argument is F_LOCK or F_TLOCK and the file is mapped with
23340 *mmap*( ).

23341 [EDEADLK] or [ENOLCK]
23342 The *function* argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request
23343 would cause the number of locks to exceed a system-imposed limit.

23344 [EOPNOTSUPP] or [EINVAL]
23345 The implementation does not support the locking of files of the type indicated
23346 by the *fildes* argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Locking a Portion of a File

In the following example, a file named **/home/cnd/mod1** is being modified. Other processes that use locking are prevented from changing it during this process. Only the first 10000 bytes are locked, and the lock call fails if another process has any part of this area locked already.

```
#include <fcntl.h>
#include <unistd.h>

int fildes;
int status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = lockf(fildes, F_TLOCK, (off_t)10000);
```

APPLICATION USAGE

Record-locking should not be used in combination with the *fopen*(), *fread*(), *fwrite*(), and other *stdio* functions. Instead, the more primitive, non-buffered functions (such as *open*()) should be used. Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The *stdio* functions are the most common source of unexpected buffering.

The *alarm*() function may be used to provide a timeout facility in applications requiring it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *chmod*(), *close*(), *creat*(), *fcntl*(), *fopen*(), *mmap*(), *open*(), *read*(), *write*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified and moved from optional to mandatory status.

A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a file that is associated with a buffered stream.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

log, logf, logl — natural logarithm function

SYNOPSIS

```
#include <math.h>

double log(double x);
float logf(float x);
long double logl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the natural logarithm of their argument x, $\log_e(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the natural logarithm of x.

If x is ±0, a pole error shall occur and *log*(), *logf*(), and *logl*() shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

MX For finite values of x that are less than 0, or if x is −Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If x is NaN, a NaN shall be returned.

If x is 1, +0 shall be returned.

If x is +Inf, x shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of x is negative, or x is −Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of x is zero.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *log10*( ), *log1p*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The *logf*( ) and *logl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

log10, log10f, log10l — base 10 logarithm function

SYNOPSIS

```
#include <math.h>

double log10(double x);
float log10f(float x);
long double log10l(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the base 10 logarithm of their argument x, $\log_{10}(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the base 10 logarithm of x.

If x is ±0, a pole error shall occur and *log10*( ), *log10f*( ), and *log10l*( ) shall return –HUGE_VAL, –HUGE_VALF, and –HUGE_VALL, respectively.

MX For finite values of x that are less than 0, or if x is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If x is NaN, a NaN shall be returned.

If x is 1, +0 shall be returned.

If x is +Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of x is negative, or x is –Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of x is zero.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *log*( ), *pow*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The *log10f*( ) and *log10l*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

log1p, log1pf, log1pl — compute a natural logarithm

SYNOPSIS

```
#include <math.h>

double log1p(double x);
float log1pf(float x);
long double log1pl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute $\log_e(1.0 + x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the natural logarithm of 1.0 + *x*.

If *x* is −1, a pole error shall occur and *log1p*( ), *log1pf*( ), and *log1pl*( ) shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

MX For finite values of *x* that are less than −1, or if *x* is −Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, or +Inf, *x* shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of *x* is less than −1, or *x* is −Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of *x* is −1.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The value of *x* is subnormal.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *log*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The *log1p*( ) function is no longer marked as an extension.

The *log1pf*( ) and *log1pl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

log2, log2f, log2l — compute base 2 logarithm functions

SYNOPSIS

```
#include <math.h>

double log2(double x);
float log2f(float x);
long double log2l(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the base 2 logarithm of their argument *x*, $\log_2(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the base 2 logarithm of *x*.

If *x* is ±0, a pole error shall occur and *log2*(), *log2f*(), and *log2l*() shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

MX For finite values of *x* that are less than 0, or if *x* is −Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is 1, +0 shall be returned.

If *x* is +Inf, *x* shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of *x* is less than zero, or *x* is −Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error — The value of *x* is zero.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

23613 **EXAMPLES**

23614 None.

23615 **APPLICATION USAGE**

23616 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
23617 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23618 **RATIONALE**

23619 None.

23620 **FUTURE DIRECTIONS**

23621 None.

23622 **SEE ALSO**

23623 *feclearexcept*( ), *fetestexcept*( ), *log*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section
23624 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

23625 **CHANGE HISTORY**

23626 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

23627 **NAME**

23628 logb, logbf, logbl — radix-independent exponent

23629 **SYNOPSIS**

23630
```
#include <math.h>
```

23631
```
double logb(double x);
```
23632
```
float logbf(float x);
```
23633
```
long double logbl(long double x);
```

23634 **DESCRIPTION**

23635 CX The functionality described on this reference page is aligned with the ISO C standard. Any
23636 conflict between the requirements described here and the ISO C standard is unintentional. This
23637 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23638 These functions shall compute the exponent of x, which is the integral part of $\log_r |x|$, as a
23639 signed floating-point value, for non-zero x, where r is the radix of the machine's floating-point
23640 arithmetic, which is the value of FLT_RADIX defined in the **<float.h>** header.

23641 If x is subnormal it is treated as though it were normalized; thus for finite positive x:

23642
$$1 <= x * \mathrm{FLT_RADIX}^{-\mathrm{logb}(x)} < \mathrm{FLT_RADIX}$$

23643 An application wishing to check for error situations should set *errno* to zero and call
23644 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
23645 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
23646 zero, an error has occurred.

23647 **RETURN VALUE**

23648 Upon successful completion, these functions shall return the exponent of x.

23649 If x is ±0, a pole error shall occur and *logb*(), *logbf*(), and *logbl*() shall return −HUGE_VAL,
23650 −HUGE_VALF, and −HUGE_VALL, respectively.

23651 MX If x is NaN, a NaN shall be returned.

23652 If x is ±Inf, +Inf shall be returned.

23653 **ERRORS**

23654 These functions shall fail if:

23655 Pole Error The value of x is ±0.

23656 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
23657 then *errno* shall be set to [ERANGE]. If the integer expression
23658 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
23659 zero floating-point exception shall be raised.

23660 **EXAMPLES**

23661 None.

23662 **APPLICATION USAGE**

23663 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
23664 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23665 **RATIONALE**

23666 None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *ilogb*( ), *scalb*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<float.h>**, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *logb*( ) function is no longer marked as an extension.

The *logbf*( ) and *logbl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

23683 **NAME**

23684 logf, logl — natural logarithm function

23685 **SYNOPSIS**

23686 `#include <math.h>`

23687 `float logf(float x);`

23688 `long double logl(long double x);`

23689 **DESCRIPTION**

23690 Refer to *log*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

longjmp — non-local goto

SYNOPSIS

```
#include <setjmp.h>

void longjmp(jmp_buf env, int val);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *longjmp*( ) function shall restore the environment saved by the most recent invocation of *setjmp*( ) in the same thread, with the corresponding **jmp_buf** argument. If there is no such invocation, or if the function containing the invocation of *setjmp*( ) has terminated execution in the interim, or if the invocation of *setjmp*( ) was within the scope of an identifier with variably CX modified type and execution has left that scope in the interim, the behavior is undefined. It is unspecified whether *longjmp*( ) restores the signal mask, leaves the signal mask unchanged, or restores it to its value at the time *setjmp*( ) was called.

All accessible objects have values, and all other components of the abstract machine have state (for example, floating-point status flags and open files), as of the time *longjmp*( ) was called, except that the values of objects of automatic storage duration are unspecified if they meet all the following conditions:

- They are local to the function containing the corresponding *setjmp*( ) invocation.
- They do not have volatile-qualified type.
- They are changed between the *setjmp*( ) invocation and *longjmp*( ) call.

CX As it bypasses the usual function call and return mechanisms, *longjmp*( ) shall execute correctly in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp*( ) is invoked from a nested signal handler (that is, from a function invoked as a result of a signal raised during the handling of another signal), the behavior is undefined.

The effect of a call to *longjmp*( ) where initialization of the **jmp_buf** structure was not performed in the calling thread is undefined.

RETURN VALUE

After *longjmp*( ) is completed, program execution continues as if the corresponding invocation of *setjmp*( ) had just returned the value specified by *val*. The *longjmp*( ) function shall not cause *setjmp*( ) to return 0; if *val* is 0, *setjmp*( ) shall return 1.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Applications whose behavior depends on the value of the signal mask should not use *longjmp*( ) and *setjmp*( ), since their effect on the signal mask is unspecified, but should instead use the *siglongjmp*( ) and *sigsetjmp*( ) functions (which can save and restore the signal mask under application control).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setjmp( ), *sigaction*( ), *siglongjmp*( ), *sigsetjmp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<setjmp.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now explicitly makes *longjmp*( )'s effect on the signal mask unspecified.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

lrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS

XSI
```
#include <stdlib.h>

long lrand48(void);
```

DESCRIPTION

Refer to *drand48*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

SYNOPSIS

```
#include <math.h>

long lrint(double x);
long lrintf(float x);
long lrintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long**, a domain error shall occur and an unspecified value is returned.

If the correct value is negative and too large to represent as a **long**, a domain error shall occur and an unspecified value is returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

23802 floating-point exception.

23803 **FUTURE DIRECTIONS**

23804 None.

23805 **SEE ALSO**

23806 *feclearexcept*( ), *fetestexcept*( ), *llrint*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
23807 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

23808 **CHANGE HISTORY**

23809 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

lround, lroundf, lroundl — round to nearest integer value

SYNOPSIS

```
#include <math.h>

long lround(double x);
long lroundf(float x);
long lroundl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is –Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long**, a domain error shall occur and an unspecified value is returned.

If the correct value is negative and too large to represent as a **long**, a domain error shall occur and an unspecified value is returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions differ from the *lrint*( ) functions in the default rounding direction, with the *lround*( ) functions rounding halfway cases away from zero and needing not to raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

23854 **FUTURE DIRECTIONS**

23855 None.

23856 **SEE ALSO**

23857 *feclearexcept*( ), *fetestexcept*( ), *llround*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
23858 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

23859 **CHANGE HISTORY**

23860 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

lsearch, lfind — linear search and update

SYNOPSIS

XSI
```
#include <search.h>

void *lsearch(const void *key, void *base, size_t *nelp, size_t width,
    int (*compar)(const void *, const void *));
void *lfind(const void *key, const void *base, size_t *nelp,
    size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

The *lsearch*( ) function shall linearly search the table and return a pointer into the table for the matching entry. If the entry does not occur, it shall be added at the end of the table. The *key* argument points to the entry to be sought in the table. The *base* argument points to the first element in the table. The *width* argument is the size of an element in bytes. The *nelp* argument points to an integer containing the current number of elements in the table. The integer to which *nelp* points shall be incremented if the entry is added to the table. The *compar* argument points to a comparison function which the application shall supply (for example, *strcmp*( )). It is called with two arguments that point to the elements being compared. The application shall ensure that the function returns 0 if the elements are equal, and non-zero otherwise.

The *lfind*( ) function shall be equivalent to *lsearch*( ), except that if the entry is not found, it is not added to the table. Instead, a null pointer is returned.

RETURN VALUE

If the searched for entry is found, both *lsearch*( ) and *lfind*( ) shall return a pointer to it. Otherwise, *lfind*( ) shall return a null pointer and *lsearch*( ) shall return a pointer to the newly added element.

Both functions shall return a null pointer in case of error.

ERRORS

No errors are defined.

EXAMPLES

Storing Strings in a Table

This fragment reads in less than or equal to TABSIZE strings of length less than or equal to ELSIZE and stores them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <string.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

...
    char line[ELSIZE], tab[TABSIZE][ELSIZE];
    size_t nel = 0;
    ...
    while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
        (void) lsearch(line, tab, &nel,
            ELSIZE, (int (*)(const void *, const void *)) strcmp);
    ...
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Finding a Matching Entry

The following example finds any line that reads "This is a test.".

```
#include <search.h>
#include <string.h>
...
char line[ELSIZE], tab[TABSIZE][ELSIZE];
size_t nel = 0;
char *findline;
void *entry;

findline = "This is a test.\n";

entry = lfind(findline, tab, &nel, ELSIZE, (
    int (*)(const void *, const void *)) strcmp);
```

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Undefined results can occur if there is not enough room in the table to add a new item.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *tsearch*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<search.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

lseek — move the read/write file offset

SYNOPSIS

```
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

DESCRIPTION

The *lseek*( ) function shall set the file offset for the open file description associated with the file descriptor *fildes*, as follows:

- If *whence* is SEEK_SET, the file offset shall be set to *offset* bytes.
- If *whence* is SEEK_CUR, the file offset shall be set to its current location plus *offset*.
- If *whence* is SEEK_END, the file offset shall be set to the size of the file plus *offset*.

The symbolic constants SEEK_SET, SEEK_CUR, and SEEK_END are defined in **<unistd.h>**.

The behavior of *lseek*( ) on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

The *lseek*( ) function shall allow the file offset to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The *lseek*( ) function shall not, by itself, extend the size of a file.

SHM If *fildes* refers to a shared memory object, the result of the *lseek*( ) function is unspecified.

TYM If *fildes* refers to a typed memory object, the result of the *lseek*( ) function is unspecified.

RETURN VALUE

Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, (**off_t**)–1 shall be returned, *errno* shall be set to indicate the error, and the file offset shall remain unchanged.

ERRORS

The *lseek*( ) function shall fail if:

[EBADF] The *fildes* argument is not an open file descriptor.

[EINVAL] The *whence* argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory.

[EOVERFLOW] The resulting file offset would be a value which cannot be represented correctly in an object of type **off_t**.

[ESPIPE] The *fildes* argument is associated with a pipe, FIFO, or socket.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard includes the functions *fgetpos*( ) and *fsetpos*( ), which work on very large files by use of a special positioning type.

Although *lseek*( ) may position the file offset beyond the end of the file, this function does not itself extend the size of the file. While the only function in IEEE Std 1003.1-2001 that may directly

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

extend the size of the file is *write*( ), *truncate*( ), and *ftruncate*( ), several functions originally derived from the ISO C standard, such as *fwrite*( ), *fprintf*( ), and so on, may do so (by causing calls on *write*( )).

An invalid file offset that would cause [EINVAL] to be returned may be both implementation-defined and device-dependent (for example, memory may have few invalid values). A negative file offset may be valid for some devices in some implementations.

The POSIX.1-1990 standard did not specifically prohibit *lseek*( ) from returning a negative offset. Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return to determine whether a return value of (**off_t**)–1 is a negative offset or an indication of an error condition. The standard developers did not wish to require this action on the part of a conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting file offset would be negative for a regular file, block special file, or directory.

FUTURE DIRECTIONS

None.

SEE ALSO

open( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EOVERFLOW] error condition is added. This change is to support large files.

An additional [ESPIPE] error condition is added for sockets.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *lseek*( ) results are unspecified for typed memory objects.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

24004 **NAME**

24005 lstat — get symbolic link status

24006 **SYNOPSIS**

24007 `#include <sys/stat.h>`

24008 `int lstat(const char *restrict path, struct stat *restrict buf);`

24009 **DESCRIPTION**

24010 The *lstat*( ) function shall be equivalent to *stat*( ), except when *path* refers to a symbolic link. In
24011 that case *lstat*( ) shall return information about the link, while *stat*( ) shall return information
24012 about the file the link references.

24013 For symbolic links, the *st_mode* member shall contain meaningful information when used with
24014 the file type macros, and the *st_size* member shall contain the length of the pathname contained
24015 in the symbolic link. File mode bits and the contents of the remaining members of the **stat**
24016 structure are unspecified. The value returned in the *st_size* member is the length of the contents
24017 of the symbolic link, and does not count any trailing null.

24018 **RETURN VALUE**

24019 Upon successful completion, *lstat*( ) shall return 0. Otherwise, it shall return –1 and set *errno* to
24020 indicate the error.

24021 **ERRORS**

24022 The *lstat*( ) function shall fail if:

24023 [EACCES] A component of the path prefix denies search permission.

24024 [EIO] An error occurred while reading from the file system.

24025 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
24026 argument.

24027 [ENAMETOOLONG]
24028 The length of a pathname exceeds {PATH_MAX} or a pathname component is
24029 longer than {NAME_MAX}.

24030 [ENOTDIR] A component of the path prefix is not a directory.

24031 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

24032 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
24033 serial number cannot be represented correctly in the structure pointed to by
24034 *buf*.

24035 The *lstat*( ) function may fail if:

24036 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
24037 resolution of the *path* argument.

24038 [ENAMETOOLONG]
24039 As a result of encountering a symbolic link in resolution of the *path* argument,
24040 the length of the substituted pathname string exceeded {PATH_MAX}.

24041 [EOVERFLOW] One of the members is too large to store into the structure pointed to by the
24042 *buf* argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

Obtaining Symbolic Link Status Information

The following example shows how to obtain status information for a symbolic link named **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path* argument specified the filename for the file pointed to by the symbolic link (**/home/cnd/mod1**), the results of calling the function would be the same as those returned by a call to the *stat*() function.

```
#include <sys/stat.h>

struct stat buffer;
int status;
...
status = lstat("/modules/pass1", &buffer);
```

APPLICATION USAGE

None.

RATIONALE

The *lstat*() function is not required to update the time-related fields if the named file is not a symbolic link. While the *st_uid*, *st_gid*, *st_atime*, *st_mtime*, and *st_ctime* members of the **stat** structure may apply to a symbolic link, they are not required to do so. No functions in IEEE Std 1003.1-2001 are required to maintain any of these time fields.

FUTURE DIRECTIONS

None.

SEE ALSO

fstat(), *readlink*(), *stat*(), *symlink*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Large File Summit extensions are added.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- This function is now mandatory.
- The [ELOOP] optional error condition is added.

The **restrict** keyword is added to the *lstat*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

makecontext, swapcontext — manipulate user contexts

SYNOPSIS

XSI
```
#include <ucontext.h>

void makecontext(ucontext_t *ucp, void (*func)(void),
    int argc, ...);
int swapcontext(ucontext_t *restrict oucp,
    const ucontext_t *restrict ucp);
```

DESCRIPTION

The *makecontext*( ) function shall modify the context specified by *ucp*, which has been initialized using *getcontext*( ). When this context is resumed using *swapcontext*( ) or *setcontext*( ), program execution shall continue by calling *func*, passing it the arguments that follow *argc* in the *makecontext*( ) call.

Before a call is made to *makecontext*( ), the application shall ensure that the context being modified has a stack allocated for it. The application shall ensure that the value of *argc* matches the number of arguments of type **int** passed to *func*; otherwise, the behavior is undefined.

The *uc_link* member is used to determine the context that shall be resumed when the context being modified by *makecontext*( ) returns. The application shall ensure that the *uc_link* member is initialized prior to the call to *makecontext*( ).

The *swapcontext*( ) function shall save the current context in the context structure pointed to by *oucp* and shall set the context to the context structure pointed to by *ucp*.

RETURN VALUE

Upon successful completion, *swapcontext*( ) shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *swapcontext*( ) function shall fail if:

[ENOMEM] The *ucp* argument does not have enough stack left to complete the operation.

EXAMPLES

The following example illustrates the use of *makecontext*( ):

```
#include <stdio.h>
#include <ucontext.h>

static ucontext_t ctx[3];

static void
f1 (void)
{
    puts("start f1");
    swapcontext(&ctx[1], &ctx[2]);
    puts("finish f1");
}

static void
f2 (void)
{
    puts("start f2");
    swapcontext(&ctx[2], &ctx[1]);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
24123           puts("finish f2");
24124       }
24125       int
24126       main (void)
24127       {
24128           char st1[8192];
24129           char st2[8192];
24130           getcontext(&ctx[1]);
24131           ctx[1].uc_stack.ss_sp = st1;
24132           ctx[1].uc_stack.ss_size = sizeof st1;
24133           ctx[1].uc_link = &ctx[0];
24134           makecontext(&ctx[1], f1, 0);
24135           getcontext(&ctx[2]);
24136           ctx[2].uc_stack.ss_sp = st2;
24137           ctx[2].uc_stack.ss_size = sizeof st2;
24138           ctx[2].uc_link = &ctx[1];
24139           makecontext(&ctx[2], f2, 0);
24140           swapcontext(&ctx[0], &ctx[2]);
24141           return 0;
24142       }
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *getcontext*( ), *sigaction*( ), *sigprocmask*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<ucontext.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the ERRORS section, the description of [ENOMEM] is changed to apply to *swapcontext*( ) only.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *swapcontext*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/33 is applied, clarifying that the arguments passed to *func* are of type **int**.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

malloc — a memory allocator

SYNOPSIS

```
#include <stdlib.h>

void *malloc(size_t size);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *malloc*( ) function shall allocate unused space for an object whose size in bytes is specified by *size* and whose value is unspecified.

The order and contiguity of storage allocated by successive calls to *malloc*( ) is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-defined: the value returned shall be either a null pointer or a unique pointer.

RETURN VALUE

Upon successful completion with *size* not equal to 0, *malloc*( ) shall return a pointer to the allocated space. If *size* is 0, either a null pointer or a unique pointer that can be successfully passed to *free*( ) shall be returned. Otherwise, it shall return a null pointer CX and set *errno* to indicate the error.

ERRORS

The *malloc*( ) function shall fail if:

CX [ENOMEM] Insufficient storage space is available.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

calloc( ), *free*( ), *realloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
- The [ENOMEM] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mblen — get number of bytes in a character

SYNOPSIS

```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If *s* is not a null pointer, *mblen*( ) shall determine the number of bytes constituting the character pointed to by *s*. Except that the shift state of *mbtowc*( ) is not affected, it shall be equivalent to:

```
mbtowc((wchar_t *)0, s, n);
```

The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *mblen*( ).

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function shall be placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes shall not produce separate wide-character codes, but shall be grouped with an adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified.

RETURN VALUE

If *s* is a null pointer, *mblen*( ) shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen*( ) shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the character (if the next *n* or fewer bytes form a valid character), or return –1 (if they do not form a valid character) CX and may set *errno* to indicate the error. In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

ERRORS

The *mblen*( ) function may fail if:

XSI [EILSEQ] Invalid character sequence is detected.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE ALSO

mbtowc( ), *mbstowcs*( ), *wctomb*( ), *wcstombs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Aligned with the ISO C standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mbrlen — get number of bytes in a character (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbrlen(const char *restrict s, size_t n,
    mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If *s* is not a null pointer, *mbrlen*( ) shall determine the number of bytes constituting the character pointed to by *s*. It shall be equivalent to:

```
mbstate_t internal;
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

If *ps* is a null pointer, the *mbrlen*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *mbrlen*( ).

The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *mbrlen*( ) function shall return the first of the following that applies:

0	If the next *n* or fewer bytes complete the character that corresponds to the null wide character.
positive	If the next *n* or fewer bytes complete a valid character; the value returned shall be the number of bytes that complete the character.
(**size_t**)–2	If the next *n* bytes contribute to an incomplete but potentially valid character, and all *n* bytes have been processed. When *n* has at least the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).
(**size_t**)–1	If an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid character. In this case, [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

ERRORS

The *mbrlen*( ) function may fail if:

[EINVAL]	*ps* points to an object that contains an invalid conversion state.
[EILSEQ]	Invalid character sequence is detected.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

24292 **EXAMPLES**

24293 None.

24294 **APPLICATION USAGE**

24295 None.

24296 **RATIONALE**

24297 None.

24298 **FUTURE DIRECTIONS**

24299 None.

24300 **SEE ALSO**

24301 *mbsinit*( ), *mbrtowc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

24302 **CHANGE HISTORY**

24303 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
24304 (E).

24305 **Issue 6**

24306 The *mbrlen*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mbrtowc — convert a character to a wide-character code (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,
    size_t n, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If *s* is a null pointer, the *mbrtowc*( ) function shall be equivalent to the call:

```
mbrtowc(NULL, "", 1, ps)
```

In this case, the values of the arguments *pwc* and *n* are ignored.

If *s* is not a null pointer, the *mbrtowc*( ) function shall inspect at most *n* bytes beginning at the byte pointed to by *s* to determine the number of bytes needed to complete the next character (including any shift sequences). If the function determines that the next character is completed, it shall determine the value of the corresponding wide character and then, if *pwc* is not a null pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide character is the null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *mbrtowc*( ) function shall use its own internal **mbstate_t** object, which shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *mbrtowc*( ).

The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *mbrtowc*( ) function shall return the first of the following that applies:

0 If the next *n* or fewer bytes complete the character that corresponds to the null wide character (which is the value stored).

between 1 and *n* inclusive
If the next *n* or fewer bytes complete a valid character (which is the value stored); the value returned shall be the number of bytes that complete the character.

(**size_t**)–2 If the next *n* bytes contribute to an incomplete but potentially valid character, and all *n* bytes have been processed (no value is stored). When *n* has at least the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).

(**size_t**)–1 If an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid character (no value is stored). In this case, [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

24349 **ERRORS**

24350 The *mbrtowc*( ) function may fail if:

24351 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

24352 [EILSEQ] Invalid character sequence is detected.

24353 **EXAMPLES**

24354 None.

24355 **APPLICATION USAGE**

24356 None.

24357 **RATIONALE**

24358 None.

24359 **FUTURE DIRECTIONS**

24360 None.

24361 **SEE ALSO**

24362 *mbsinit*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

24363 **CHANGE HISTORY**

24364 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
24365 (E).

24366 **Issue 6**

24367 The *mbrtowc*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24368 The following new requirements on POSIX implementations derive from alignment with the
24369 Single UNIX Specification:

24370 • The [EINVAL] error condition is added.

24371 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mbsinit — determine conversion object status

SYNOPSIS

```
#include <wchar.h>

int mbsinit(const mbstate_t *ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If *ps* is not a null pointer, the *mbsinit*( ) function shall determine whether the object pointed to by *ps* describes an initial conversion state.

RETURN VALUE

The *mbsinit*( ) function shall return non-zero if *ps* is a null pointer, or if the pointed-to object describes an initial conversion state; otherwise, it shall return zero.

If an **mbstate_t** object is altered by any of the functions described as ''restartable'', and is then used with a different character sequence, or in the other conversion direction, or with a different *LC_CTYPE* category setting than on earlier function calls, the behavior is undefined.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The **mbstate_t** object is used to describe the current conversion state from a particular character sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the *LC_CTYPE* category of the current locale.

The initial conversion state corresponds, for a conversion in either direction, to the beginning of a new character sequence in the initial shift state. A zero valued **mbstate_t** object is at least one way to describe an initial conversion state. A zero valued **mbstate_t** object can be used to initiate conversion involving any character sequence, in any *LC_CTYPE* category setting.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbrlen( ), *mbrtowc*( ), *wcrtomb*( ), *mbsrtowcs*( ), *wcsrtombs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mbsrtowcs — convert a character string to a wide-character string (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
    size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *mbsrtowcs*( ) function shall convert a sequence of characters, beginning in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters shall be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which shall also be stored. Conversion shall stop early in either of the following cases:

- A sequence of bytes is encountered that does not form a valid character.
- *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).

Each conversion shall take place as if by a call to the *mbrtowc*( ) function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last character converted (if any). If conversion stopped due to reaching a terminating null character, and if *dst* is not a null pointer, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *mbsrtowcs*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation behaves as if no function defined in this volume of IEEE Std 1003.1-2001 calls *mbsrtowcs*( ).

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

If the input conversion encounters a sequence of bytes that do not form a valid character, an encoding error occurs. In this case, the *mbsrtowcs*( ) function stores the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)–1; the conversion state is undefined. Otherwise, it shall return the number of characters successfully converted, not including the terminating null (if any).

ERRORS

The *mbsrtowcs*( ) function may fail if:

CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

[EILSEQ] Invalid character sequence is detected.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *mbrtowc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The *mbsrtowcs*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

The [EINVAL] error condition is marked CX.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mbstowcs — convert a character string to a wide-character string

SYNOPSIS

```
#include <stdlib.h>

size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *mbstowcs*( ) function shall convert a sequence of characters that begins in the initial shift state from the array pointed to by *s* into a sequence of corresponding wide-character codes and shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No characters that follow a null byte (which is converted into a wide-character code with value 0) shall be examined or converted. Each character shall be converted as if by a call to *mbtowc*( ), except that the shift state of *mbtowc*( ) is not affected.

No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes place between objects that overlap, the behavior is undefined.

XSI The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale. If *pwcs* is a null pointer, *mbstowcs*( ) shall return the length required to convert the entire array regardless of the value of *n*, but no values are stored.

RETURN VALUE

CX If an invalid character is encountered, *mbstowcs*( ) shall return (**size_t**)–1 and may set *errno* to indicate the error.

XSI Otherwise, *mbstowcs*( ) shall return the number of the array elements modified (or required if *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if the value returned is *n*.

ERRORS

The *mbstowcs*( ) function may fail if:

XSI [EILSEQ] Invalid byte sequence is detected.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen( ), *mbtowc*( ), *wctomb*( ), *wcstombs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

24508 **CHANGE HISTORY**

24509 First released in Issue 4. Aligned with the ISO C standard.

24510 **Issue 6**

24511 The *mbstowcs*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

24512 Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

mbtowc — convert a character to a wide-character code

SYNOPSIS

```
#include <stdlib.h>

int mbtowc(wchar_t *restrict pwc, const char *restrict s, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If *s* is not a null pointer, *mbtowc*( ) shall determine the number of bytes that constitute the character pointed to by *s*. It shall then determine the wide-character code for the value of type **wchar_t** that corresponds to that character. (The value of the wide-character code corresponding to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc*( ) shall store the wide-character code in the object pointed to by *pwc*.

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function is placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes shall not produce separate wide-character codes, but shall be grouped with an adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *mbtowc*( ).

RETURN VALUE

If *s* is a null pointer, *mbtowc*( ) shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc*( ) shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the CX converted character (if the next *n* or fewer bytes form a valid character), or return −1 and may set *errno* to indicate the error(if they do not form a valid character).

In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

ERRORS

The *mbtowc*( ) function may fail if:

XSI [EILSEQ] Invalid character sequence is detected.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE ALSO

mblen( ), *mbstowcs*( ), *wctomb*( ), *wcstombs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4. Aligned with the ISO C standard.

Issue 6

The *mbtowc*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

memccpy — copy bytes in memory

SYNOPSIS

XSI

```
#include <string.h>

void *memccpy(void *restrict s1, const void *restrict s2,
    int c, size_t n);
```

DESCRIPTION

The *memccpy*( ) function shall copy bytes from memory area *s2* into *s1*, stopping after the first occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied, whichever comes first. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *memccpy*( ) function shall return a pointer to the byte after the copy of *c* in *s1*, or a null pointer if *c* was not found in the first *n* bytes of *s2*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *memccpy*( ) function does not check for the overflow of the receiving memory area.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The **restrict** keyword is added to the *memccpy*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

memchr — find byte in memory

SYNOPSIS

```
#include <string.h>

void *memchr(const void *s, int c, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *memchr*( ) function shall locate the first occurrence of *c* (converted to an **unsigned char**) in the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

RETURN VALUE

The *memchr*( ) function shall return a pointer to the located byte, or a null pointer if the byte does not occur in the object.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

memcmp — compare bytes in memory

SYNOPSIS

```
#include <string.h>

int memcmp(const void *s1, const void *s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *memcmp*( ) function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects being compared.

RETURN VALUE

The *memcmp*( ) function shall return an integer greater than, equal to, or less than 0, if the object pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

24655 **NAME**

24656 memcpy — copy bytes in memory

24657 **SYNOPSIS**

24658 `#include <string.h>`

24659 `void *memcpy(void *restrict s1, const void *restrict s2, size_t n);`

24660 **DESCRIPTION**

24661 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24662 conflict between the requirements described here and the ISO C standard is unintentional. This
24663 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24664 The *memcpy*( ) function shall copy *n* bytes from the object pointed to by *s2* into the object pointed
24665 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

24666 **RETURN VALUE**

24667 The *memcpy*( ) function shall return *s1*; no return value is reserved to indicate an error.

24668 **ERRORS**

24669 No errors are defined.

24670 **EXAMPLES**

24671 None.

24672 **APPLICATION USAGE**

24673 The *memcpy*( ) function does not check for the overflow of the receiving memory area.

24674 **RATIONALE**

24675 None.

24676 **FUTURE DIRECTIONS**

24677 None.

24678 **SEE ALSO**

24679 The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

24680 **CHANGE HISTORY**

24681 First released in Issue 1. Derived from Issue 1 of the SVID.

24682 **Issue 6**

24683 The *memcpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

memmove — copy bytes in memory with overlapping areas

SYNOPSIS

```
#include <string.h>

void *memmove(void *s1, const void *s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *memmove*( ) function shall copy *n* bytes from the object pointed to by *s2* into the object pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

RETURN VALUE

The *memmove*( ) function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

24713 **NAME**

24714 memset — set bytes in memory

24715 **SYNOPSIS**

24716 `#include <string.h>`

24717 `void *memset(void *s, int c, size_t n);`

24718 **DESCRIPTION**

24719 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24720 conflict between the requirements described here and the ISO C standard is unintentional. This
24721 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24722 The *memset*( ) function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes
24723 of the object pointed to by *s*.

24724 **RETURN VALUE**

24725 The *memset*( ) function shall return *s*; no return value is reserved to indicate an error.

24726 **ERRORS**

24727 No errors are defined.

24728 **EXAMPLES**

24729 None.

24730 **APPLICATION USAGE**

24731 None.

24732 **RATIONALE**

24733 None.

24734 **FUTURE DIRECTIONS**

24735 None.

24736 **SEE ALSO**

24737 The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

24738 **CHANGE HISTORY**

24739 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mkdir — make a directory

SYNOPSIS

```
#include <sys/stat.h>

int mkdir(const char *path, mode_t mode);
```

DESCRIPTION

The *mkdir*( ) function shall create a new directory with name *path*. The file permission bits of the new directory shall be initialized from *mode*. These file permission bits of the *mode* argument shall be modified by the process' file creation mask.

When bits in *mode* other than the file permission bits are set, the meaning of these additional bits is implementation-defined.

The directory's user ID shall be set to the process' effective user ID. The directory's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the directory's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the directory's group ID to the effective group ID of the calling process.

The newly created directory shall be an empty directory.

If *path* names a symbolic link, *mkdir*( ) shall fail and set *errno* to [EEXIST].

Upon successful completion, *mkdir*( ) shall mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the directory. Also, the *st_ctime* and *st_mtime* fields of the directory that contains the new entry shall be marked for update.

RETURN VALUE

Upon successful completion, *mkdir*( ) shall return 0. Otherwise, –1 shall be returned, no directory shall be created, and *errno* shall be set to indicate the error.

ERRORS

The *mkdir*( ) function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.

[EEXIST] The named file exists.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[EMLINK] The link count of the parent directory would exceed {LINK_MAX}.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of the path prefix specified by *path* does not name an existing directory or *path* is an empty string.

[ENOSPC] The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.

[ENOTDIR] A component of the path prefix is not a directory.

[EROFS] The parent directory resides on a read-only file system.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *mkdir*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

EXAMPLES

Creating a Directory

The following example shows how to create a directory named **/home/cnd/mod1**, with read/write/search permissions for owner and group, and with read/search permissions for others.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
...
status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

APPLICATION USAGE

None.

RATIONALE

The *mkdir*( ) function originated in 4.2 BSD and was added to System V in Release 3.0.

4.3 BSD detects [ENAMETOOLONG].

The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*( ) to set the group ID after the directory is created, or determine under what conditions the implementation will set the desired group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

umask( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mkfifo — make a FIFO special file

SYNOPSIS

```
#include <sys/stat.h>

int mkfifo(const char *path, mode_t mode);
```

DESCRIPTION

The *mkfifo*( ) function shall create a new FIFO special file named by the pathname pointed to by *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission bits of the *mode* argument shall be modified by the process' file creation mask.

When bits in *mode* other than the file permission bits are set, the effect is implementation-defined.

If *path* names a symbolic link, *mkfifo*( ) shall fail and set *errno* to [EEXIST].

The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the FIFO's group ID to the effective group ID of the calling process.

Upon successful completion, *mkfifo*( ) shall mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file. Also, the *st_ctime* and *st_mtime* fields of the directory that contains the new entry shall be marked for update.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned, no FIFO shall be created, and *errno* shall be set to indicate the error.

ERRORS

The *mkfifo*( ) function shall fail if:

[EACCES] A component of the path prefix denies search permission, or write permission is denied on the parent directory of the FIFO to be created.

[EEXIST] The named file already exists.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of the path prefix specified by *path* does not name an existing directory or *path* is an empty string.

[ENOSPC] The directory that would contain the new file cannot be extended or the file system is out of file-allocation resources.

[ENOTDIR] A component of the path prefix is not a directory.

[EROFS] The named file resides on a read-only file system.

The *mkfifo*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

EXAMPLES

Creating a FIFO File

The following example shows how to create a FIFO file named **/home/cnd/mod_done**, with read/write permissions for owner, and with read permissions for group and others.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
...
status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
    S_IRGRP | S_IROTH);
```

APPLICATION USAGE

None.

RATIONALE

The syntax of this function is intended to maintain compatibility with historical implementations of *mknod*(). The latter function was included in the 1984 /usr/group standard but only for use in creating FIFO special files. The *mknod*() function was originally excluded from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir*() and *mkfifo*(). The *mknod*() function is now included for alignment with the Single UNIX Specification.

The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*() to set the group ID after the FIFO is created, or determine under what conditions the implementation will set the desired group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

umask(), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

mknod — make a directory, a special file, or a regular file

SYNOPSIS

XSI
```
#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev);
```

DESCRIPTION

The *mknod*( ) function shall create a new file named by the pathname to which the argument *path* points.

The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the following symbolic constants:

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

The only portable use of *mknod*( ) is to create a FIFO-special file. If *mode* is not S_IFIFO or *dev* is not 0, the behavior of *mknod*( ) is unspecified.

The permissions for the new file are OR'ed into the *mode* argument, and may be selected from any combination of the following symbolic constants:

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

The user ID of the file shall be initialized to the effective user ID of the process. The group ID of the file shall be initialized to either the effective group ID of the process or the group ID of the parent directory. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process. The owner, group, and other permission bits of *mode* shall be modified by the file mode creation mask of the process. The *mknod*( ) function shall clear each bit whose corresponding bit in the file mode creation mask of the process is set.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If *path* names a symbolic link, *mknod*( ) shall fail and set *errno* to [EEXIST].

Upon successful completion, *mknod*( ) shall mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file. Also, the *st_ctime* and *st_mtime* fields of the directory that contains the new entry shall be marked for update.

Only a process with appropriate privileges may invoke *mknod*( ) for file types other than FIFO-special.

RETURN VALUE

Upon successful completion, *mknod*( ) shall return 0. Otherwise, it shall return –1, the new file shall not be created, and *errno* shall be set to indicate the error.

ERRORS

The *mknod*( ) function shall fail if:

[EACCES] A component of the path prefix denies search permission, or write permission is denied on the parent directory.

[EEXIST] The named file exists.

[EINVAL] An invalid argument exists.

[EIO] An I/O error occurred while accessing the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of the path prefix specified by *path* does not name an existing directory or *path* is an empty string.

[ENOSPC] The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.

[ENOTDIR] A component of the path prefix is not a directory.

[EPERM] The invoking process does not have appropriate privileges and the file type is not FIFO-special.

[EROFS] The directory in which the file is to be created is located on a read-only file system.

The *mknod*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Creating a FIFO Special File

The following example shows how to create a FIFO special file named **/home/cnd/mod_done**, with read/write permissions for owner, and with read permissions for group and others.

```
#include <sys/types.h>
#include <sys/stat.h>

dev_t dev;
int   status;
...
status  = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
    S_IRUSR | S_IRGRP | S_IROTH, dev);
```

APPLICATION USAGE

The *mkfifo*( ) function is preferred over this function for making FIFO special files.

RATIONALE

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*( ) to set the group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *creat*( ), *exec*, *mkdir*( ), *mkfifo*( ), *open*( ), *stat*( ), *umask*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mkstemp — make a unique filename

SYNOPSIS

XSI
```
#include <stdlib.h>

int mkstemp(char *template);
```

DESCRIPTION

The *mkstemp*( ) function shall replace the contents of the string pointed to by *template* by a unique filename, and return a file descriptor for the file open for reading and writing. The function thus prevents any possible race condition between testing whether the file exists and opening it for use. The string in *template* should look like a filename with six trailing 'X's; *mkstemp*( ) replaces each 'X' with a character from the portable filename character set. The characters are chosen such that the resulting name does not duplicate the name of an existing file at the time of a call to *mkstemp*( ).

RETURN VALUE

Upon successful completion, *mkstemp*( ) shall return an open file descriptor. Otherwise, –1 shall be returned if no suitable file could be created.

ERRORS

No errors are defined.

EXAMPLES

Generating a Filename

The following example creates a file with a 10-character name beginning with the characters "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file descriptor that identifies the file.

```
#include <stdlib.h>
...
char template[] = "/tmp/fileXXXXXX";
int fd;

fd = mkstemp(template);
```

APPLICATION USAGE

It is possible to run out of letters.

The *mkstemp*( ) function need not check to determine whether the filename part of *template* exceeds the maximum allowable filename length.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid( ), *open*( ), *tmpfile*( ), *tmpnam*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

25074 **CHANGE HISTORY**

25075 First released in Issue 4, Version 2.

25076 **Issue 5**

25077 Moved from X/OPEN UNIX extension to BASE.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

25078 **NAME**

25079 mktemp — make a unique filename (**LEGACY**)

25080 **SYNOPSIS**

25081 XSI
```
#include <stdlib.h>

char *mktemp(char *template);
```
25082

25083

25084 **DESCRIPTION**

25085 The *mktemp*( ) function shall replace the contents of the string pointed to by *template* by a unique
25086 filename and return *template*. The application shall initialize *template* to be a filename with six
25087 trailing 'X's; *mktemp*( ) shall replace each 'X' with a single byte character from the portable
25088 filename character set.

25089 **RETURN VALUE**

25090 The *mktemp*( ) function shall return the pointer *template*. If a unique name cannot be created,
25091 *template* shall point to a null string.

25092 **ERRORS**

25093 No errors are defined.

25094 **EXAMPLES**

25095 **Generating a Filename**

25096 The following example replaces the contents of the "template" string with a 10-character
25097 filename beginning with the characters "file" and returns a pointer to the "template" string
25098 that contains the new filename.

```
25099 #include <stdlib.h>
25100 ...
25101 char *template = "/tmp/fileXXXXXX";
25102 char *ptr;

25103 ptr = mktemp(template);
```

25104 **APPLICATION USAGE**

25105 Between the time a pathname is created and the file opened, it is possible for some other process
25106 to create a file with the same name. The *mkstemp*( ) function avoids this problem and is preferred
25107 over this function.

25108 **RATIONALE**

25109 None.

25110 **FUTURE DIRECTIONS**

25111 This function may be withdrawn in a future version.

25112 **SEE ALSO**

25113 *mkstemp*( ), *tmpfile*( ), *tmpnam*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

25114 **CHANGE HISTORY**

25115 First released in Issue 4, Version 2.

25116 **Issue 5**

25117 Moved from X/OPEN UNIX extension to BASE.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

25118 **Issue 6**

25119 This function is marked LEGACY.

25120 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mktime — convert broken-down time into time since the Epoch

SYNOPSIS

```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *mktime*( ) function shall convert the broken-down time, expressed as local time, in the structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that of the values returned by *time*( ). The original values of the *tm_wday* and *tm_yday* components of the structure are ignored, and the original values of the other components are not restricted to the ranges described in **<time.h>**.

CX A positive or 0 value for *tm_isdst* shall cause *mktime*( ) to presume initially that Daylight Savings Time, respectively, is or is not in effect for the specified time. A negative value for *tm_isdst* shall cause *mktime*( ) to attempt to determine whether Daylight Savings Time is in effect for the specified time.

Local timezone information shall be set as though *mktime*( ) called *tzset*( ).

The relationship between the **tm** structure (defined in the **<time.h>** header) and the time in seconds since the Epoch is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch) corrected for timezone and any seasonal time adjustments, where the names in the structure and in the expression correspond.

Upon successful completion, the values of the *tm_wday* and *tm_yday* components of the structure shall be set appropriately, and the other components are set to represent the specified time since the Epoch, but with their values forced to the ranges indicated in the **<time.h>** entry; the final value of *tm_mday* shall not be set until *tm_mon* and *tm_year* are determined.

RETURN VALUE

The *mktime*( ) function shall return the specified time since the Epoch encoded as a value of type **time_t**. If the time since the Epoch cannot be represented, the function shall return the value (**time_t**)–1.

ERRORS

No errors are defined.

EXAMPLES

What day of the week is July 4, 2001?

```
#include <stdio.h>
#include <time.h>

struct tm time_str;

char daybuf[20];

int main(void)
{
    time_str.tm_year = 2001 - 1900;
    time_str.tm_mon = 7 - 1;
    time_str.tm_mday = 4;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            time_str.tm_hour = 0;
            time_str.tm_min = 0;
            time_str.tm_sec = 1;
            time_str.tm_isdst = -1;
            if (mktime(&time_str) == -1)
                (void)puts("-unknown-");
            else {
                (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
                (void)puts(daybuf);
            }
            return 0;
        }
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25192 **NAME**

25193 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)

25194 **SYNOPSIS**

25195 MLR
```
#include <sys/mman.h>

int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
```
25196

25197

25198

25199 **DESCRIPTION**

25200 The *mlock*( ) function shall cause those whole pages containing any part of the address space of
25201 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until
25202 unlocked or until the process exits or *exec*s another process image. The implementation may
25203 require that *addr* be a multiple of {PAGESIZE}.

25204 The *munlock*( ) function shall unlock those whole pages containing any part of the address space
25205 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many
25206 times *mlock*( ) has been called by the process for any of the pages in the specified range. The
25207 implementation may require that *addr* be a multiple of {PAGESIZE}.

25208 If any of the pages in the range specified to a call to *munlock*( ) are also mapped into the address
25209 spaces of other processes, any locks established on those pages by another process are
25210 unaffected by the call of this process to *munlock*( ). If any of the pages in the range specified by a
25211 call to *munlock*( ) are also mapped into other portions of the address space of the calling process
25212 outside the range specified, any locks established on those pages via the other mappings are also
25213 unaffected by this call.

25214 Upon successful return from *mlock*( ), pages in the specified range shall be locked and memory-
25215 resident. Upon successful return from *munlock*( ), pages in the specified range shall be unlocked
25216 with respect to the address space of the process. Memory residency of unlocked pages is
25217 unspecified.

25218 The appropriate privilege is required to lock process memory with *mlock*( ).

25219 **RETURN VALUE**

25220 Upon successful completion, the *mlock*( ) and *munlock*( ) functions shall return a value of zero.
25221 Otherwise, no change is made to any locks in the address space of the process, and the function
25222 shall return a value of –1 and set *errno* to indicate the error.

25223 **ERRORS**

25224 The *mlock*( ) and *munlock*( ) functions shall fail if:

25225 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does
25226 not correspond to valid mapped pages in the address space of the process.

25227 The *mlock*( ) function shall fail if:

25228 [EAGAIN] Some or all of the memory identified by the operation could not be locked
25229 when the call was made.

25230 The *mlock*( ) and *munlock*( ) functions may fail if:

25231 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

25232 The *mlock*( ) function may fail if:

25233 [ENOMEM] Locking the pages mapped by the specified range would exceed an
25234 implementation-defined limit on the amount of memory that the process may
25235 lock.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

25236 [EPERM] The calling process does not have the appropriate privilege to perform the
25237 requested operation.

25238 **EXAMPLES**

25239 None.

25240 **APPLICATION USAGE**

25241 None.

25242 **RATIONALE**

25243 None.

25244 **FUTURE DIRECTIONS**

25245 None.

25246 **SEE ALSO**

25247 *exec*, *exit*( ), *fork*( ), *mlockall*( ), *munmap*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
25248 **<sys/mman.h>**

25249 **CHANGE HISTORY**

25250 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25251 **Issue 6**

25252 The *mlock*( ) and *munlock*( ) functions are marked as part of the Range Memory Locking option.

25253 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25254 implementation does not support the Range Memory Locking option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)

SYNOPSIS

ML
```
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);
```

DESCRIPTION

The *mlockall*( ) function shall cause all of the pages mapped by the address space of a process to be memory-resident until unlocked or until the process exits or *exec*s another process image. The *flags* argument determines whether the pages to be locked are those currently mapped by the address space of the process, those that are mapped in the future, or both. The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following symbolic constants, defined in **<sys/mman.h>**:

MCL_CURRENT Lock all of the pages currently mapped into the address space of the process.

MCL_FUTURE Lock all of the pages that become mapped into the address space of the process in the future, when those mappings are established.

If MCL_FUTURE is specified, and the automatic locking of future mappings eventually causes the amount of locked memory to exceed the amount of available physical memory or any other implementation-defined limit, the behavior is implementation-defined. The manner in which the implementation informs the application of these situations is also implementation-defined.

The *munlockall*( ) function shall unlock all currently mapped pages of the address space of the process. Any pages that become mapped into the address space of the process after a call to *munlockall*( ) shall not be locked, unless there is an intervening call to *mlockall*( ) specifying MCL_FUTURE or a subsequent call to *mlockall*( ) specifying MCL_CURRENT. If pages mapped into the address space of the process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by a call by this process to *munlockall*( ).

Upon successful return from the *mlockall*( ) function that specifies MCL_CURRENT, all currently mapped pages of the process' address space shall be memory-resident and locked. Upon return from the *munlockall*( ) function, all currently mapped pages of the process' address space shall be unlocked with respect to the process' address space. The memory residency of unlocked pages is unspecified.

The appropriate privilege is required to lock process memory with *mlockall*( ).

RETURN VALUE

Upon successful completion, the *mlockall*( ) function shall return a value of zero. Otherwise, no additional memory shall be locked, and the function shall return a value of –1 and set *errno* to indicate the error. The effect of failure of *mlockall*( ) on previously existing locks in the address space is unspecified.

If it is supported by the implementation, the *munlockall*( ) function shall always return a value of zero. Otherwise, the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *mlockall*( ) function shall fail if:

[EAGAIN] Some or all of the memory identified by the operation could not be locked when the call was made.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

25300 [EINVAL] The *flags* argument is zero, or includes unimplemented flags.

25301 The *mlockall*( ) function may fail if:

25302 [ENOMEM] Locking all of the pages currently mapped into the address space of the
25303 process would exceed an implementation-defined limit on the amount of
25304 memory that the process may lock.

25305 [EPERM] The calling process does not have the appropriate privilege to perform the
25306 requested operation.

25307 **EXAMPLES**

25308 None.

25309 **APPLICATION USAGE**

25310 None.

25311 **RATIONALE**

25312 None.

25313 **FUTURE DIRECTIONS**

25314 None.

25315 **SEE ALSO**

25316 *exec*, *exit*( ), *fork*( ), *mlock*( ), *munmap*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
25317 **<sys/mman.h>**

25318 **CHANGE HISTORY**

25319 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25320 **Issue 6**

25321 The *mlockall*( ) and *munlockall*( ) functions are marked as part of the Process Memory Locking
25322 option.

25323 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25324 implementation does not support the Process Memory Locking option.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mmap — map pages of memory

SYNOPSIS

MC3
```
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags,
    int fildes, off_t off);
```

DESCRIPTION

The *mmap*( ) function shall establish a mapping between a process' address space and a file, shared memory object, or typed memory object. The format of the call is as follows:

```
pa=mmap(addr, len, prot, flags, fildes, off);
```

The *mmap*( ) function shall establish a mapping between the address space of the process at an address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and the values of *flags*, further described below. A successful *mmap*( ) call shall return *pa* as its result. The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) address space of the process. The range of bytes starting at *off* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the file, shared memory object, or typed memory object represented by *fildes*.

TYM If *fildes* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, the memory object to be mapped shall be that portion of the typed memory object allocated by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap*( ) is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling process, *mmap*( ) shall fail.

The mapping established by *mmap*( ) shall replace any previous mappings for those whole pages containing any part of the address space of the process starting at *pa* and continuing for *len* bytes.

If the size of the mapped file changes after the call to *mmap*( ) as a result of some other operation on the mapped file, the effect of references to portions of the mapped region that correspond to added or removed portions of the file is unspecified.

TYM The *mmap*( ) function shall be supported for regular files, shared memory objects, and typed memory objects. Support for any other type of file is unspecified.

The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The *prot* shall be either PROT_NONE or the bitwise-inclusive OR of one or more of the other flags in the following table, defined in the **<sys/mman.h>** header.

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap*( ) shall fail.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

MPR An implementation may permit accesses other than those specified by *prot*; however, if the Memory Protection option is supported, the implementation shall not permit a write to succeed where PROT_WRITE has not been set or shall not permit any access where PROT_NONE alone has been set. The implementation shall support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and PROT_WRITE. If the Memory Protection option is not supported, the result of any access that conflicts with the specified protection is undefined. The file descriptor *fildes* shall have been opened with read permission, regardless of the protection options specified. If PROT_WRITE is specified, the application shall ensure that it has opened the file descriptor *fildes* with write permission unless MAP_PRIVATE is specified in the *flags* parameter as described below.

The parameter *flags* provides other information about the handling of the mapped data. The value of *flags* is the bitwise-inclusive OR of these options, defined in **<sys/mman.h>**:

Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret *addr* exactly.

Implementations that do not support the Memory Mapped Files option are not required to support MAP_PRIVATE.

XSI It is implementation-defined whether MAP_FIXED shall be supported. MAP_FIXED shall be supported on XSI-conformant systems.

MAP_SHARED and MAP_PRIVATE describe the disposition of write references to the memory object. If MAP_SHARED is specified, write references shall change the underlying object. If MAP_PRIVATE is specified, modifications to the mapped data by the calling process shall be visible only to the calling process and shall not change the underlying object. It is unspecified whether modifications to the underlying object done after the MAP_PRIVATE mapping is established are visible through the MAP_PRIVATE mapping. Either MAP_SHARED or MAP_PRIVATE can be specified, but not both. The mapping type is retained across *fork*().

TYM When *fildes* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, *mmap*() shall, if there are enough resources available, map *len* bytes allocated from the corresponding typed memory object which were not previously allocated to any process in any processor that may access that typed memory object. If there are not enough resources available, the function shall fail. If *fildes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, these allocated bytes shall be contiguous within the typed memory object. If *fildes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE flag, these allocated bytes may be composed of non-contiguous fragments within the typed memory object. If *fildes* represents a typed memory object opened with neither the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag nor the POSIX_TYPED_MEM_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory object are mapped, exactly as when mapping a file or shared memory object. In this case, if two processes map an area of typed memory using the same *off* and *len* values and using file descriptors that refer to the same memory pool (either from the same port or from a different port), both processes shall map the same region of storage.

When MAP_FIXED is set in the *flags* argument, the implementation is informed that the value of *pa* shall be *addr*, exactly. If MAP_FIXED is set, *mmap*() may return MAP_FAILED and set *errno* to [EINVAL]. If a MAP_FIXED request is successful, the mapping established by *mmap*() replaces any previous mappings for the process' pages in the range [*pa*,*pa*+*len*).

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When MAP_FIXED is not set, the implementation uses *addr* in an implementation-defined manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the implementation deems suitable for a mapping of *len* bytes to the file. All implementations interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the implementation selects a value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

The *off* argument is constrained to be aligned and sized according to the value returned by *sysconf*( ) when passed _SC_PAGESIZE or _SC_PAGE_SIZE. When MAP_FIXED is specified, the application shall ensure that the argument *addr* also meets these constraints. The implementation performs mapping operations over whole pages. Thus, while the argument *len* need not meet a size or alignment constraint, the implementation shall include, in any mapping operation, any partial page specified by the range [*pa*,*pa*+*len*).

The system shall always zero-fill any partial page at the end of an object. Further, the system shall never write out any modified portions of the last page of an object which are beyond its end. MPR References within the address range starting at *pa* and continuing for *len* bytes to whole pages following the end of an object shall result in delivery of a SIGBUS signal.

An implementation may generate SIGBUS signals when a reference would cause an error in the mapped object, such as out-of-space condition.

The *mmap*( ) function shall add an extra reference to the file associated with the file descriptor *fildes* which is not removed by a subsequent *close*( ) on that file descriptor. This reference shall be removed when there are no more mappings to the file.

The *st_atime* field of the mapped file may be marked for update at any time between the *mmap*( ) call and the corresponding *munmap*( ) call. The initial read or write reference to a mapped region shall cause the file's *st_atime* field to be marked for update if it has not already been marked for update.

The *st_ctime* and *st_mtime* fields of a file that is mapped with MAP_SHARED and PROT_WRITE shall be marked for update at some point in the interval between a write reference to the mapped region and the next call to *msync*( ) with MS_ASYNC or MS_SYNC for that portion of the file by any process. If there is no such call and if the underlying file is modified as a result of a write reference, then these fields shall be marked for update at some time after the write reference.

There may be implementation-defined limits on the number of memory regions that can be mapped (per process or per system).

XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a process is decreased by the use of *shmat*( ) is implementation-defined.

If *mmap*( ) fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the mappings in the address range starting at *addr* and continuing for *len* bytes may have been unmapped.

RETURN VALUE

Upon successful completion, the *mmap*( ) function shall return the address at which the mapping was placed (*pa*); otherwise, it shall return a value of MAP_FAILED and set *errno* to indicate the error. The symbol MAP_FAILED is defined in the **<sys/mman.h>** header. No successful return from *mmap*( ) shall return the value MAP_FAILED.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

25462 **ERRORS**

25463 The *mmap*( ) function shall fail if:

25464 [EACCES] The *fildes* argument is not open for read, regardless of the protection specified,
25465 or *fildes* is not open for write and PROT_WRITE was specified for a
25466 MAP_SHARED type mapping.

25467 ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall*( ), due to
25468 a lack of resources.

25469 [EBADF] The *fildes* argument is not a valid open file descriptor.

25470 [EINVAL] The *addr* argument (if MAP_FIXED was specified) or *off* is not a multiple of
25471 the page size as returned by *sysconf*( ), or is considered invalid by the
25472 implementation.

25473 [EINVAL] The value of *flags* is invalid (neither MAP_PRIVATE nor MAP_SHARED is
25474 set).

25475 [EMFILE] The number of mapped regions would exceed an implementation-defined
25476 limit (per process or per system).

25477 [ENODEV] The *fildes* argument refers to a file whose type is not supported by *mmap*( ).

25478 [ENOMEM] MAP_FIXED was specified, and the range [*addr*,*addr*+*len*) exceeds that allowed
25479 for the address space of a process; or, if MAP_FIXED was not specified and
25480 there is insufficient room in the address space to effect the mapping.

25481 ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall*( ),
25482 because it would require more space than the system is able to supply.

25483 TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory
25484 object designated by *fildes* to allocate *len* bytes.

25485 [ENOTSUP] MAP_FIXED or MAP_PRIVATE was specified in the *flags* argument and the
25486 implementation does not support this functionality.

25487 The implementation does not support the combination of accesses requested
25488 in the *prot* argument.

25489 [ENXIO] Addresses in the range [*off*,*off*+*len*) are invalid for the object specified by *fildes*.

25490 [ENXIO] MAP_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is
25491 invalid for the object specified by *fildes*.

25492 TYM [ENXIO] The *fildes* argument refers to a typed memory object that is not accessible from
25493 the calling process.

25494 [EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset
25495 maximum established in the open file description associated with *fildes*.

25496 **EXAMPLES**

25497 None.

25498 **APPLICATION USAGE**

25499 Use of *mmap*( ) may reduce the amount of memory available to other memory allocation
25500 functions.

25501 Use of MAP_FIXED may result in unspecified behavior in further use of *malloc*( ) and *shmat*( ).
25502 The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the
25503 most effective use of resources.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The application must ensure correct synchronization when using *mmap*( ) in conjunction with any other file access method, such as *read*( ) and *write*( ), standard input/output, and *shmat*( ).

The *mmap*( ) function allows access to resources via address space manipulations, instead of *read*( )/*write*( ). Once a file is mapped, all a process has to do to access it is use the data at the address to which the file was mapped. So, using pseudo-code to illustrate the way in which an existing program might be changed to use *mmap*( ), the following:

```
fildes = open(...)
lseek(fildes, some_offset)
read(fildes, buf, len)
/* Use data in buf. */
```

becomes:

```
fildes = open(...)
address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
/* Use data at address. */
```

RATIONALE

After considering several other alternatives, it was decided to adopt the *mmap*( ) definition found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is minimal, in that it describes only what has been built, and what appears to be necessary for a general and portable mapping facility.

Note that while *mmap*( ) was first designed for mapping files, it is actually a general-purpose mapping facility. It can be used to map any appropriate object, such as memory, files, devices, and so on, into the address space of a process.

When a mapping is established, it is possible that the implementation may need to map more than is requested into the address space of the process because of hardware requirements. An application, however, cannot count on this behavior. Implementations that do not use a paged architecture may simply allocate a common memory region and return the address of it; such implementations probably do not allocate any more than is necessary. References past the end of the requested area are unspecified.

If an application requests a mapping that would overlay existing mappings in the process, it might be desirable that an implementation detect this and inform the application. However, the default, portable (not MAP_FIXED) operation does not overlay existing mappings. On the other hand, if the program specifies a fixed address mapping (which requires some implementation knowledge to determine a suitable address, if the function is supported at all), then the program is presumed to be successfully managing its own address space and should be trusted when it asks to map over existing data structures. Furthermore, it is also desirable to make as few system calls as possible, and it might be considered onerous to require an *munmap*( ) before an *mmap*( ) to the same address range. This volume of IEEE Std 1003.1-2001 specifies that the new mappings replace any existing mappings, following existing practice in this regard.

It is not expected, when the Memory Protection option is supported, that all hardware implementations are able to support all combinations of permissions at all addresses. When this option is supported, implementations are required to disallow write access to mappings without write permission and to disallow access to mappings without any access permission. Other than these restrictions, implementations may allow access types other than those requested by the application. For example, if the application requests only PROT_WRITE, the implementation may also allow read access. A call to *mmap*( ) fails if the implementation cannot support allowing all the access requested by the application. For example, some implementations cannot support a request for both write access and execute access simultaneously. All implementations supporting the Memory Protection option must support requests for no access, read access,

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

write access, and both read and write access. Strictly conforming code must only rely on the required checks. These restrictions allow for portability across a wide range of hardware.

The MAP_FIXED address treatment is likely to fail for non-page-aligned values and for certain architecture-dependent address ranges. Conforming implementations cannot count on being able to choose address values for MAP_FIXED without utilizing non-portable, implementation-defined knowledge. Nonetheless, MAP_FIXED is provided as a standard interface conforming to existing practice for utilizing such knowledge when it is available.

Similarly, in order to allow implementations that do not support virtual addresses, support for directly specifying any mapping addresses via MAP_FIXED is not required and thus a conforming application may not count on it.

The MAP_PRIVATE function can be implemented efficiently when memory protection hardware is available. When such hardware is not available, implementations can implement such ''mappings'' by simply making a real copy of the relevant data into process private memory, though this tends to behave similarly to *read*( ).

The function has been defined to allow for many different models of using shared memory. However, all uses are not equally portable across all machine architectures. In particular, the *mmap*( ) function allows the system as well as the application to specify the address at which to map a specific region of a memory object. The most portable way to use the function is always to let the system choose the address, specifying NULL as the value for the argument *addr* and not to specify MAP_FIXED.

If it is intended that a particular region of a memory object be mapped at the same address in a group of processes (on machines where this is even possible), then MAP_FIXED can be used to pass in the desired mapping address. The system can still be used to choose the desired address if the first such mapping is made without specifying MAP_FIXED, and then the resulting mapping address can be passed to subsequent processes for them to pass in via MAP_FIXED. The availability of a specific address range cannot be guaranteed, in general.

The *mmap*( ) function can be used to map a region of memory that is larger than the current size of the object. Memory access within the mapping but beyond the current end of the underlying objects may result in SIGBUS signals being sent to the process. The reason for this is that the size of the object can be manipulated by other processes and can change at any moment. The implementation should tell the application that a memory reference is outside the object where this can be detected; otherwise, written data may be lost and read data may not reflect actual data in the object.

Note that references beyond the end of the object do not extend the object as the new end cannot be determined precisely by most virtual memory hardware. Instead, the size can be directly manipulated by *ftruncate*( ).

Process memory locking does apply to shared memory regions, and the MEMLOCK_FUTURE argument to *mlockall*( ) can be relied upon to cause new shared memory regions to be automatically locked.

Existing implementations of *mmap*( ) return the value –1 when unsuccessful. Since the casting of this value to type **void *** cannot be guaranteed by the ISO C standard to be distinct from a successful value, this volume of IEEE Std 1003.1-2001 defines the symbol MAP_FAILED, which a conforming implementation does not return as the result of a successful call.

FUTURE DIRECTIONS

None.

--`,,`-`-`,,`,,`,`,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25597 **SEE ALSO**

25598 *exec*, *fcntl*( ), *fork*( ), *lockf*( ), *msync*( ), *munmap*( ), *mprotect*( ), *posix_typed_mem_open*( ), *shmat*( ),
25599 *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/mman.h>**

25600 **CHANGE HISTORY**

25601 First released in Issue 4, Version 2.

25602 **Issue 5**

25603 Moved from X/OPEN UNIX extension to BASE.

25604 Aligned with *mmap*( ) in the POSIX Realtime Extension as follows:

25605 • The DESCRIPTION is extensively reworded.

25606 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.

25607 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.

25608 • The value returned on failure is the value of the constant MAP_FAILED; this was previously
25609 defined as –1.

25610 Large File Summit extensions are added.

25611 **Issue 6**

25612 The *mmap*( ) function is marked as part of the Memory Mapped Files option.

25613 The Open Group Corrigendum U028/6 is applied, changing (**void** *)–1 to MAP_FAILED.

25614 The following new requirements on POSIX implementations derive from alignment with the
25615 Single UNIX Specification:

25616 • The DESCRIPTION is updated to describe the use of MAP_FIXED.

25617 • The DESCRIPTION is updated to describe the addition of an extra reference to the file
25618 associated with the file descriptor passed to *mmap*( ).

25619 • The DESCRIPTION is updated to state that there may be implementation-defined limits on
25620 the number of memory regions that can be mapped.

25621 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*
25622 argument.

25623 • The [EINVAL] and [EMFILE] error conditions are added.

25624 • The [EOVERFLOW] error condition is added. This change is to support large files.

25625 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

25626 • The DESCRIPTION is updated to describe the cases when MAP_PRIVATE and MAP_FIXED
25627 need not be supported.

25628 The following changes are made for alignment with IEEE Std 1003.1j-2000:

25629 • Semantics for typed memory objects are added to the DESCRIPTION.

25630 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.

25631 • The *posix_typed_mem_open*( ) function is added to the SEE ALSO section.

25632 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

25633 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code |
25634 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM). |

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

25635 **NAME**

25636 modf, modff, modfl — decompose a floating-point number

25637 **SYNOPSIS**

25638
```
#include <math.h>
```

25639
```
double modf(double x, double *iptr);
```
25640
```
float modff(float value, float *iptr);
```
25641
```
long double modfl(long double value, long double *iptr);
```

25642 **DESCRIPTION**

25643 CX The functionality described on this reference page is aligned with the ISO C standard. Any
25644 conflict between the requirements described here and the ISO C standard is unintentional. This
25645 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

25646 These functions shall break the argument *x* into integral and fractional parts, each of which has
25647 the same sign as the argument. It stores the integral part as a **double** (for the *modf*( ) function), a
25648 **float** (for the *modff*( ) function), or a **long double** (for the *modfl*( ) function), in the object pointed
25649 to by *iptr*.

25650 **RETURN VALUE**

25651 Upon successful completion, these functions shall return the signed fractional part of *x*.

25652 MX If *x* is NaN, a NaN shall be returned, and **iptr* shall be set to a NaN.

25653 If *x* is ±Inf, ±0 shall be returned, and **iptr* shall be set to ±Inf.

25654 **ERRORS**

25655 No errors are defined.

25656 **EXAMPLES**

25657 None.

25658 **APPLICATION USAGE**

25659 The *modf*( ) function computes the function result and **iptr* such that:

25660
```
a = modf(x, iptr) ;
```
25661
```
x == a+*iptr ;
```

25662 allowing for the usual floating-point inaccuracies.

25663 **RATIONALE**

25664 None.

25665 **FUTURE DIRECTIONS**

25666 None.

25667 **SEE ALSO**

25668 *frexp*( ), *isnan*( ), *ldexp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

25669 **CHANGE HISTORY**

25670 First released in Issue 1. Derived from Issue 1 of the SVID.

25671 **Issue 5**

25672 The DESCRIPTION is updated to indicate how an application should check for an error. This
25673 text was previously published in the APPLICATION USAGE section.

25674 **Issue 6**

25675 The *modff*( ) and *modfl*( ) functions are added for alignment with the ISO/IEC 9899:1999
25676 standard.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example in the APPLICATION USAGE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mprotect — set protection of memory mapping

SYNOPSIS

MPR

```
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);
```

DESCRIPTION

The *mprotect*( ) function shall change the access protections to be that specified by *prot* for those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The *prot* argument should be either PROT_NONE or the bitwise-inclusive OR of one or more of PROT_READ, PROT_WRITE, and PROT_EXEC.

If an implementation cannot support the combination of access types specified by *prot*, the call to *mprotect*( ) shall fail.

An implementation may permit accesses other than those specified by *prot*; however, no implementation shall permit a write to succeed where PROT_WRITE has not been set or shall permit any access where PROT_NONE alone has been set. Implementations shall support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and PROT_WRITE. If PROT_WRITE is specified, the application shall ensure that it has opened the mapped objects in the specified address range with write permission, unless MAP_PRIVATE was specified in the original mapping, regardless of whether the file descriptors used to map the objects have since been closed.

The implementation shall require that *addr* be a multiple of the page size as returned by *sysconf*( ).

The behavior of this function is unspecified if the mapping was not established by a call to *mmap*( ).

When *mprotect*( ) fails for reasons other than [EINVAL], the protections on some of the pages in the range [*addr*,*addr*+*len*) may have been changed.

RETURN VALUE

Upon successful completion, *mprotect*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *mprotect*( ) function shall fail if:

[EACCES] The *prot* argument specifies a protection that violates the access permission the process has to the underlying memory object.

[EAGAIN] The *prot* argument specifies PROT_WRITE over a MAP_PRIVATE mapping and there are insufficient memory resources to reserve for locking the private page.

[EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf*( ).

[ENOMEM] Addresses in the range [*addr*,*addr*+*len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

[ENOMEM] The *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and it would require more space than the system is able to supply for locking the private pages, if required.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25728 [ENOTSUP] The implementation does not support the combination of accesses requested
25729 in the *prot* argument.

25730 **EXAMPLES**

25731 None.

25732 **APPLICATION USAGE**

25733 The [EINVAL] error above is marked EX because it is defined as an optional error in the POSIX
25734 Realtime Extension.

25735 **RATIONALE**

25736 None.

25737 **FUTURE DIRECTIONS**

25738 None.

25739 **SEE ALSO**

25740 *mmap*( ), *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/mman.h>**

25741 **CHANGE HISTORY**

25742 First released in Issue 4, Version 2.

25743 **Issue 5**

25744 Moved from X/OPEN UNIX extension to BASE.

25745 Aligned with *mprotect*( ) in the POSIX Realtime Extension as follows:

25746 • The DESCRIPTION is largely reworded.

25747 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.

25748 • [EAGAIN] is moved from the optional to the mandatory error conditions.

25749 **Issue 6**

25750 The *mprotect*( ) function is marked as part of the Memory Protection option.

25751 The following new requirements on POSIX implementations derive from alignment with the
25752 Single UNIX Specification:

25753 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of
25754 the page size as returned by *sysconf*( ).

25755 • The [EINVAL] error condition is added.

25756 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mq_close — close a message queue (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

DESCRIPTION

The *mq_close*( ) function shall remove the association between the message queue descriptor, *mqdes*, and its message queue. The results of using this message queue descriptor after successful return from this *mq_close*( ), and until the return of this message queue descriptor from a subsequent *mq_open*( ), are undefined.

If the process has successfully attached a notification request to the message queue via this *mqdes*, this attachment shall be removed, and the message queue is available for another process to attach for notification.

RETURN VALUE

Upon successful completion, the *mq_close*( ) function shall return a value of zero; otherwise, the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *mq_close*( ) function shall fail if:

[EBADF] The *mqdes* argument is not a valid message queue descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_close*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25794 **NAME**

25795 mq_getattr — get message queue attributes (**REALTIME**)

25796 **SYNOPSIS**

25797 MSG
```
#include <mqueue.h>
```

25798
```
int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

25799

25800 **DESCRIPTION**

25801 The *mq_getattr*( ) function shall obtain status information and attributes of the message queue
25802 and the open message queue description associated with the message queue descriptor.

25803 The *mqdes* argument specifies a message queue descriptor.

25804 The results shall be returned in the **mq_attr** structure referenced by the *mqstat* argument.

25805 Upon return, the following members shall have the values associated with the open message
25806 queue description as set when the message queue was opened and as modified by subsequent
25807 *mq_setattr*( ) calls: *mq_flags*.

25808 The following attributes of the message queue shall be returned as set at message queue
25809 creation: *mq_maxmsg*, *mq_msgsize*.

25810 Upon return, the following members within the **mq_attr** structure referenced by the *mqstat*
25811 argument shall be set to the current state of the message queue:

25812 *mq_curmsgs* The number of messages currently on the queue.

25813 **RETURN VALUE**

25814 Upon successful completion, the *mq_getattr*( ) function shall return zero. Otherwise, the function
25815 shall return –1 and set *errno* to indicate the error.

25816 **ERRORS**

25817 The *mq_getattr*( ) function shall fail if:

25818 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

25819 **EXAMPLES**

25820 None.

25821 **APPLICATION USAGE**

25822 None.

25823 **RATIONALE**

25824 None.

25825 **FUTURE DIRECTIONS**

25826 None.

25827 **SEE ALSO**

25828 *mq_open*( ), *mq_send*( ), *mq_setattr*( ), *mq_timedsend*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), the
25829 Base Definitions volume of IEEE Std 1003.1-2001, **<mqueue.h>**

25830 **CHANGE HISTORY**

25831 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25832 **Issue 6**

25833 The *mq_getattr*( ) function is marked as part of the Message Passing option.

25834 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25835 implementation does not support the Message Passing option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *mq_timedsend*() function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25838 **NAME**

25839 mq_notify — notify process that a message is available (**REALTIME**)

25840 **SYNOPSIS**

25841 MSG
```
#include <mqueue.h>

int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```
25842

25843

25844 **DESCRIPTION**

25845 If the argument *notification* is not NULL, this function shall register the calling process to be
25846 notified of message arrival at an empty message queue associated with the specified message
25847 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to
25848 the process when the message queue transitions from empty to non-empty. At any time, only
25849 one process may be registered for notification by a message queue. If the calling process or any
25850 other process has already registered for notification of message arrival at the specified message
25851 queue, subsequent attempts to register for that message queue shall fail.

25852 If *notification* is NULL and the process is currently registered for notification by the specified
25853 message queue, the existing registration shall be removed.

25854 When the notification is sent to the registered process, its registration shall be removed. The
25855 message queue shall then be available for registration.

25856 If a process has registered for notification of message arrival at a message queue and some
25857 thread is blocked in *mq_receive*( ) waiting to receive a message when a message arrives at the
25858 queue, the arriving message shall satisfy the appropriate *mq_receive*( ). The resulting behavior is
25859 as if the message queue remains empty, and no notification shall be sent.

25860 **RETURN VALUE**

25861 Upon successful completion, the *mq_notify*( ) function shall return a value of zero; otherwise, the
25862 function shall return a value of –1 and set *errno* to indicate the error.

25863 **ERRORS**

25864 The *mq_notify*( ) function shall fail if:

25865 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

25866 [EBUSY] A process is already registered for notification by the message queue.

25867 **EXAMPLES**

25868 None.

25869 **APPLICATION USAGE**

25870 None.

25871 **RATIONALE**

25872 None.

25873 **FUTURE DIRECTIONS**

25874 None.

25875 **SEE ALSO**

25876 *mq_open*( ), *mq_send*( ), *mq_timedsend*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), the Base
25877 Definitions volume of IEEE Std 1003.1-2001, **<mqueue.h>**

25878 **CHANGE HISTORY**

25879 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

The *mq_notify*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25886 **NAME**

25887 mq_open — open a message queue (**REALTIME**)

25888 **SYNOPSIS**

25889 MSG
```
#include <mqueue.h>
```

25890
```
mqd_t mq_open(const char *name, int oflag, ...);
```

25891

25892 **DESCRIPTION**

25893 The *mq_open*( ) function shall establish the connection between a process and a message queue
25894 with a message queue descriptor. It shall create an open message queue description that refers to
25895 the message queue, and a message queue descriptor that refers to that open message queue
25896 description. The message queue descriptor is used by other functions to refer to that message
25897 queue. The *name* argument points to a string naming a message queue. It is unspecified whether
25898 the name appears in the file system and is visible to other functions that take pathnames as
25899 arguments. The *name* argument shall conform to the construction rules for a pathname. If *name*
25900 begins with the slash character, then processes calling *mq_open*( ) with the same value of *name*
25901 shall refer to the same message queue object, as long as that name has not been removed. If *name*
25902 does not begin with the slash character, the effect is implementation-defined. The interpretation
25903 of slash characters other than the leading slash character in *name* is implementation-defined. If
25904 the *name* argument is not the name of an existing message queue and creation is not requested,
25905 *mq_open*( ) shall fail and return an error.

25906 A message queue descriptor may be implemented using a file descriptor, in which case
25907 applications can open up to at least {OPEN_MAX} file and message queues.

25908 The *oflag* argument requests the desired receive and/or send access to the message queue. The
25909 requested access permission to receive messages or send messages shall be granted if the calling
25910 process would be granted read or write access, respectively, to an equivalently protected file.

25911 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications
25912 shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

25913 O_RDONLY Open the message queue for receiving messages. The process can use the
25914 returned message queue descriptor with *mq_receive*( ), but not *mq_send*( ). A
25915 message queue may be open multiple times in the same or different processes
25916 for receiving messages.

25917 O_WRONLY Open the queue for sending messages. The process can use the returned
25918 message queue descriptor with *mq_send*( ) but not *mq_receive*( ). A message
25919 queue may be open multiple times in the same or different processes for
25920 sending messages.

25921 O_RDWR Open the queue for both receiving and sending messages. The process can use
25922 any of the functions allowed for O_RDONLY and O_WRONLY. A message
25923 queue may be open multiple times in the same or different processes for
25924 sending messages.

25925 Any combination of the remaining flags may be specified in the value of *oflag*:

25926 O_CREAT Create a message queue. It requires two additional arguments: *mode*, which
25927 shall be of type **mode_t**, and *attr*, which shall be a pointer to an **mq_attr**
25928 structure. If the pathname *name* has already been used to create a message
25929 queue that still exists, then this flag shall have no effect, except as noted under
25930 O_EXCL. Otherwise, a message queue shall be created without any messages
25931 in it. The user ID of the message queue shall be set to the effective user ID of
25932 the process, and the group ID of the message queue shall be set to the effective

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

group ID of the process. The file permission bits shall be set to the value of *mode*. When bits in *mode* other than file permission bits are set, the effect is implementation-defined. If *attr* is NULL, the message queue shall be created with implementation-defined default message queue attributes. If *attr* is non-NULL and the calling process has the appropriate privilege on *name*, the message queue *mq_maxmsg* and *mq_msgsize* attributes shall be set to the values of the corresponding members in the **mq_attr** structure referred to by *attr*. If *attr* is non-NULL, but the calling process does not have the appropriate privilege on *name*, the *mq_open*( ) function shall fail and return an error without creating the message queue.

O_EXCL If O_EXCL and O_CREAT are set, *mq_open*( ) shall fail if the message queue *name* exists. The check for the existence of the message queue and the creation of the message queue if it does not exist shall be atomic with respect to other threads executing *mq_open*( ) naming the same *name* with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_NONBLOCK Determines whether an *mq_send*( ) or *mq_receive*( ) waits for resources or messages that are not currently available, or fails with *errno* set to [EAGAIN]; see *mq_send*( ) and *mq_receive*( ) for details.

The *mq_open*( ) function does not add or remove messages from the queue.

RETURN VALUE

Upon successful completion, the function shall return a message queue descriptor; otherwise, the function shall return (**mqd_t**)–1 and set *errno* to indicate the error.

ERRORS

The *mq_open*( ) function shall fail if:

[EACCES] The message queue exists and the permissions specified by *oflag* are denied, or the message queue does not exist and permission to create the message queue is denied.

[EEXIST] O_CREAT and O_EXCL are set and the named message queue already exists.

[EINTR] The *mq_open*( ) function was interrupted by a signal.

[EINVAL] The *mq_open*( ) function is not supported for the given name.

[EINVAL] O_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either *mq_maxmsg* or *mq_msgsize* was less than or equal to zero.

[EMFILE] Too many message queue descriptors or file descriptors are currently in use by this process.

[ENAMETOOLONG] The length of the *name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENFILE] Too many message queues are currently open in the system.

[ENOENT] O_CREAT is not set and the named message queue does not exist.

[ENOSPC] There is insufficient space for the creation of the new message queue.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25974 **EXAMPLES**

25975 None.

25976 **APPLICATION USAGE**

25977 None.

25978 **RATIONALE**

25979 None.

25980 **FUTURE DIRECTIONS**

25981 None.

25982 **SEE ALSO**

25983 *mq_close*( ), *mq_getattr*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_timedreceive*( ), *mq_timedsend*( ),
25984 *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), the Base Definitions volume of
25985 IEEE Std 1003.1-2001, **<mqueue.h>**

25986 **CHANGE HISTORY**

25987 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25988 **Issue 6**

25989 The *mq_open*( ) function is marked as part of the Message Passing option.

25990 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25991 implementation does not support the Message Passing option.

25992 The *mq_timedreceive*( ) and *mq_timedsend*( ) functions are added to the SEE ALSO section for
25993 alignment with IEEE Std 1003.1d-1999.

25994 The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mq_receive, mq_timedreceive — receive a message from a message queue (**REALTIME**)

SYNOPSIS

MSG
```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
    unsigned *msg_prio);
```

MSG TMO
```
#include <mqueue.h>
#include <time.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
    size_t msg_len, unsigned *restrict msg_prio,
    const struct timespec *restrict abs_timeout);
```

DESCRIPTION

The *mq_receive*( ) function shall receive the oldest of the highest priority message(s) from the message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg_len* argument, is less than the *mq_msgsize* attribute of the message queue, the function shall fail and return an error. Otherwise, the selected message shall be removed from the queue and copied to the buffer pointed to by the *msg_ptr* argument.

If the value of *msg_len* is greater than {SSIZE_MAX}, the result is implementation-defined.

If the argument *msg_prio* is not NULL, the priority of the selected message shall be stored in the location referenced by *msg_prio*.

If the specified message queue is empty and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_receive*( ) shall block until a message is enqueued on the message queue or until *mq_receive*( ) is interrupted by a signal. If more than one thread is waiting to receive a message when a message arrives at an empty queue and the Priority Scheduling option is supported, then the thread of highest priority that has been waiting the longest shall be selected to receive the message. Otherwise, it is unspecified which waiting thread receives the message. If the specified message queue is empty and O_NONBLOCK is set in the message queue description associated with *mqdes*, no message shall be removed from the queue, and *mq_receive*( ) shall return an error.

TMO The *mq_timedreceive*( ) function shall receive the oldest of the highest priority messages from the message queue specified by *mqdes* as described for the *mq_receive*( ) function. However, if O_NONBLOCK was not specified when the message queue was opened via the *mq_open*( ) function, and no message exists on the queue to satisfy the receive, the wait for such a message shall be terminated when the specified timeout expires. If O_NONBLOCK is set, this function is equivalent to *mq_receive*( ).

The timeout expires when the absolute time specified by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout* has already been passed at the time of the call.

TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock; if the Timers option is not supported, the timeout shall be based on the system clock as returned by the *time*( ) function.

TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The *timespec* argument is defined in the **<time.h>** header.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Under no circumstance shall the operation fail with a timeout if a message can be removed from the message queue immediately. The validity of the *abs_timeout* parameter need not be checked if a message can be removed from the message queue immediately.

RETURN VALUE

TMO Upon successful completion, the *mq_receive*( ) and *mq_timedreceive*( ) functions shall return the length of the selected message in bytes and the message shall be removed from the queue. Otherwise, no message shall be removed from the queue, the functions shall return a value of –1, and set *errno* to indicate the error.

ERRORS

TMO The *mq_receive*( ) and *mq_timedreceive*( ) functions shall fail if:

[EAGAIN] O_NONBLOCK was set in the message description associated with *mqdes*, and the specified message queue is empty.

[EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.

[EMSGSIZE] The specified message buffer size, *msg_len*, is less than the message size attribute of the message queue.

TMO [EINTR] The *mq_receive*( ) or *mq_timedreceive*( ) operation was interrupted by a signal.

TMO [EINVAL] The process or thread would have blocked, and the *abs_timeout* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

TMO [ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened, but no message arrived on the queue before the specified timeout expired.

TMO The *mq_receive*( ) and *mq_timedreceive*( ) functions may fail if:

[EBADMSG] The implementation has detected a data corruption problem with the message.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_send*( ), *mq_timedsend*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), *time*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<mqueue.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_receive*( ) function is marked as part of the Message Passing option.

The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to *msg_len* rather than *maxsize*.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The *mq_timedreceive*( ) function is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *mq_timedreceive*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq_timedreceive*( ) from **int** to **ssize_t**.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mq_send, mq_timedsend — send a message to a message queue (**REALTIME**)

SYNOPSIS

MSG
```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio);
```

MSG TMO
```
#include <mqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio, const struct timespec *abs_timeout);
```

DESCRIPTION

The *mq_send*( ) function shall add the message pointed to by the argument *msg_ptr* to the message queue specified by *mqdes*. The *msg_len* argument specifies the length of the message, in bytes, pointed to by *msg_ptr*. The value of *msg_len* shall be less than or equal to the *mq_msgsize* attribute of the message queue, or *mq_send*( ) shall fail.

If the specified message queue is not full, *mq_send*( ) shall behave as if the message is inserted into the message queue at the position indicated by the *msg_prio* argument. A message with a larger numeric value of *msg_prio* shall be inserted before messages with lower values of *msg_prio*. A message shall be inserted after other messages in the queue, if any, with equal *msg_prio*. The value of *msg_prio* shall be less than {MQ_PRIO_MAX}.

If the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_send*( ) shall block until space becomes available to enqueue the message, or until *mq_send*( ) is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue and the Priority Scheduling option is supported, then the thread of the highest priority that has been waiting the longest shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and O_NONBLOCK is set in the message queue description associated with *mqdes*, the message shall not be queued and *mq_send*( ) shall return an error.

TMO The *mq_timedsend*( ) function shall add a message to the message queue specified by *mqdes* in the manner defined for the *mq_send*( ) function. However, if the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for sufficient room in the queue shall be terminated when the specified timeout expires. If O_NONBLOCK is set in the message queue description, this function shall be equivalent to *mq_send*( ).

The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout* has already been passed at the time of the call.

TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock; if the Timers option is not supported, the timeout shall be based on the system clock as returned by the *time*( ) function.

TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The *timespec* argument is defined in the **<time.h>** header.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Under no circumstance shall the operation fail with a timeout if there is sufficient room in the queue to add the message immediately. The validity of the *abs_timeout* parameter need not be checked when there is sufficient room in the queue.

RETURN VALUE

TMO Upon successful completion, the *mq_send*( ) and *mq_timedsend*( ) functions shall return a value of zero. Otherwise, no message shall be enqueued, the functions shall return –1, and *errno* shall be set to indicate the error.

ERRORS

TMO The *mq_send*( ) and *mq_timedsend*( ) functions shall fail if:

[EAGAIN] The O_NONBLOCK flag is set in the message queue description associated with *mqdes*, and the specified message queue is full.

[EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.

TMO [EINTR] A signal interrupted the call to *mq_send*( ) or *mq_timedsend*( ).

[EINVAL] The value of *msg_prio* was outside the valid range.

TMO [EINVAL] The process or thread would have blocked, and the *abs_timeout* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

[EMSGSIZE] The specified message length, *msg_len*, exceeds the message size attribute of the message queue.

TMO [ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened, but the timeout expired before the message could be added to the queue.

EXAMPLES

None.

APPLICATION USAGE

The value of the symbol {MQ_PRIO_MAX} limits the number of priority levels supported by the application. Message priorities range from 0 to {MQ_PRIO_MAX}–1.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_receive*( ), *mq_setattr*( ), *mq_timedreceive*( ), *time*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<mqueue.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_send*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend*( ) function is added for alignment with IEEE Std 1003.1d-1999.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

mq_setattr — set message queue attributes (**REALTIME**)

SYNOPSIS

MSG

```
#include <mqueue.h>

int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
    struct mq_attr *restrict omqstat);
```

DESCRIPTION

The *mq_setattr*( ) function shall set attributes associated with the open message queue description referenced by the message queue descriptor specified by *mqdes*.

The message queue attributes corresponding to the following members defined in the **mq_attr** structure shall be set to the specified values upon successful completion of *mq_setattr*( ):

mq_flags The value of this member is the bitwise-logical OR of zero or more of O_NONBLOCK and any implementation-defined flags.

The values of the *mq_maxmsg*, *mq_msgsize*, and *mq_curmsgs* members of the **mq_attr** structure shall be ignored by *mq_setattr*( ).

If *omqstat* is non-NULL, the *mq_setattr*( ) function shall store, in the location referenced by *omqstat*, the previous message queue attributes and the current queue status. These values shall be the same as would be returned by a call to *mq_getattr*( ) at that point.

RETURN VALUE

Upon successful completion, the function shall return a value of zero and the attributes of the message queue shall have been changed as specified.

Otherwise, the message queue attributes shall be unchanged, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_setattr*( ) function shall fail if:

[EBADF] The *mqdes* argument is not a valid message queue descriptor.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_send*( ), *mq_timedsend*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 6

The *mq_setattr*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *mq_setattr*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

26228 **NAME**

26229 mq_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)

26230 **SYNOPSIS**

26231 MSG TMO
```
#include <mqueue.h>
#include <time.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
    size_t msg_len, unsigned *restrict msg_prio,
    const struct timespec *restrict abs_timeout);
```
26236

26237 **DESCRIPTION**

26238 Refer to *mq_receive*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

mq_timedsend — send a message to a message queue (**ADVANCED REALTIME**)

SYNOPSIS

MSG TMO

```
#include <mqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio, const struct timespec *abs_timeout);
```

DESCRIPTION

Refer to *mq_send*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mq_unlink — remove a message queue (**REALTIME**)

SYNOPSIS

MSG
```
#include <mqueue.h>

int mq_unlink(const char *name);
```

DESCRIPTION

The *mq_unlink*( ) function shall remove the message queue named by the pathname *name*. After a successful call to *mq_unlink*( ) with *name*, a call to *mq_open*( ) with *name* shall fail if the flag O_CREAT is not set in *flags*. If one or more processes have the message queue open when *mq_unlink*( ) is called, destruction of the message queue shall be postponed until all references to the message queue have been closed.

Calls to *mq_open*( ) to recreate the message queue may fail until the message queue is actually removed. However, the *mq_unlink*( ) call need not block until all references have been closed; it may return immediately.

RETURN VALUE

Upon successful completion, the function shall return a value of zero. Otherwise, the named message queue shall be unchanged by this function call, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *mq_unlink*( ) function shall fail if:

[EACCES] Permission is denied to unlink the named message queue.

[ENAMETOOLONG]
The length of the *name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] The named message queue does not exist.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_close( ), *mq_open*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<mqueue.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_unlink*( ) function is marked as part of the Message Passing option.

The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion, the named message queue is unchanged by this function.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

26292 The [ENOSYS] error condition has been removed as stubs need not be provided if an
26293 implementation does not support the Message Passing option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

mrand48 — generate uniformly distributed pseudo-random signed long integers

SYNOPSIS

XSI

```
#include <stdlib.h>

long mrand48(void);
```

DESCRIPTION

Refer to *drand48*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

msgctl — XSI message control operations

SYNOPSIS

XSI
```
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

DESCRIPTION

The *msgctl*( ) function operates on XSI message queues (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *msgctl*( ) function shall provide message control operations as specified by *cmd*. The following values for *cmd*, and the message control operations they specify, are:

IPC_STAT
: Place the current value of each member of the **msqid_ds** data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in **<sys/msg.h>**.

IPC_SET
: Set the value of the following members of the **msqid_ds** data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:

  `msg_perm.uid`
  `msg_perm.gid`
  `msg_perm.mode`
  `msg_qbytes`

  IPC_SET can only be executed by a process with appropriate privileges or that has an effective user ID equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*. Only a process with appropriate privileges can raise the value of **msg_qbytes**.

IPC_RMID
: Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and **msqid_ds** data structure associated with it. IPC_RMD can only be executed by a process with appropriate privileges or one that has an effective user ID equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*.

RETURN VALUE

Upon successful completion, *msgctl*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *msgctl*( ) function shall fail if:

[EACCES]
: The argument *cmd* is IPC_STAT and the calling process does not have read permission; see Section 2.7 (on page 39).

[EINVAL]
: The value of *msqid* is not a valid message queue identifier; or the value of *cmd* is not a valid command.

[EPERM]
: The argument *cmd* is IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[EPERM] The argument *cmd* is IPC_SET, an attempt is being made to increase to the value of **msg_qbytes**, and the effective user ID of the calling process does not have appropriate privileges.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/msg.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

msgget — get the XSI message queue identifier

SYNOPSIS

XSI
```
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

DESCRIPTION

The *msgget*( ) function operates on XSI message queues (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *msgget*( ) function shall return the message queue identifier associated with the argument *key*.

A message queue identifier, associated message queue, and data structure (see **<sys/msg.h>**), shall be created for the argument *key* if one of the following is true:

- The argument *key* is equal to IPC_PRIVATE.
- The argument *key* does not already have a message queue identifier associated with it, and (*msgflg* & IPC_CREAT) is non-zero.

Upon creation, the data structure associated with the new message queue identifier shall be initialized as follows:

- **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** shall be set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **msg_perm.mode** shall be set equal to the low-order 9 bits of *msgflg*.
- **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** shall be set equal to 0.
- **msg_ctime** shall be set equal to the current time.
- **msg_qbytes** shall be set equal to the system limit.

RETURN VALUE

Upon successful completion, *msgget*( ) shall return a non-negative integer, namely a message queue identifier. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *msgget*( ) function shall fail if:

[EACCES] A message queue identifier exists for the argument *key*, but operation permission as specified by the low-order 9 bits of *msgflg* would not be granted; see Section 2.7 (on page 39).

[EEXIST] A message queue identifier exists for the argument *key* but ((*msgflg* & IPC_CREAT) && (*msgflg* & IPC_EXCL)) is non-zero.

[ENOENT] A message queue identifier does not exist for the argument *key* and (*msgflg* & IPC_CREAT) is 0.

[ENOSPC] A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgrcv*( ), *msgsnd*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/msg.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

msgrcv — XSI message receive operation

SYNOPSIS

XSI
```
#include <sys/msg.h>

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
    int msgflg);
```

DESCRIPTION

The *msgrcv*( ) function operates on XSI message queues (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *msgrcv*( ) function shall read a message from the queue associated with the message queue identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long    mtype;     /* Message type. */
    char    mtext[1];  /* Message text. */
}
```

The structure member *mtype* is the received message's type as specified by the sending process.

The structure member *mtext* is the text of the message.

The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG_NOERROR) is non-zero. The truncated part of the message shall be lost and no indication of the truncation shall be given to the calling process.

If the value of *msgsz* is greater than {SSIZE_MAX}, the result is implementation-defined.

The argument *msgtyp* specifies the type of message requested as follows:

- If *msgtyp* is 0, the first message on the queue shall be received.
- If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.
- If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* shall be received.

The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

- If (*msgflg* & IPC_NOWAIT) is non-zero, the calling thread shall return immediately with a return value of –1 and *errno* set to [ENOMSG].
- If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:
  - — A message of the desired type is placed on the queue.
  - — The message queue identifier *msqid* is removed from the system; when this occurs, *errno* shall be set equal to [EIDRM] and –1 shall be returned.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— The calling thread receives a signal that is to be caught; in this case a message is not received and the calling thread resumes execution in the manner prescribed in *sigaction*( ).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*:

- **msg_qnum** shall be decremented by 1.
- **msg_lrpid** shall be set equal to the process ID of the calling process.
- **msg_rtime** shall be set equal to the current time.

RETURN VALUE

Upon successful completion, *msgrcv*( ) shall return a value equal to the number of bytes actually placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv*( ) shall return (**ssize_t**)–1, and *errno* shall be set to indicate the error.

ERRORS

The *msgrcv*( ) function shall fail if:

[E2BIG]	The value of *mtext* is greater than *msgsz* and (*msgflg* & MSG_NOERROR) is 0.
[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 39).
[EIDRM]	The message queue identifier *msqid* is removed from the system.
[EINTR]	The *msgrcv*( ) function was interrupted by a signal.
[EINVAL]	*msqid* is not a valid message queue identifier.
[ENOMSG]	The queue does not contain a message of the desired type and (*msgflg* & IPC_NOWAIT) is non-zero.

EXAMPLES

Receiving a Message

The following example receives the first message on the queue (based on the value of the *msgtyp* argument, 0). The queue is identified by the *msqid* argument (assuming that the value has previously been set). This call specifies that an error should be reported if no message is available, but not if the message is too large. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
    long type;
    char text[20];
} msg;
long msgtyp = 0;
...
result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
         msgtyp, MSG_NOERROR | IPC_NOWAIT);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgsnd*( ), *sigaction*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/msg.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The type of the return value is changed from **int** to **ssize_t**, and a warning is added to the DESCRIPTION about values of *msgsz* larger the {SSIZE_MAX}.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to the APPLICATION USAGE section.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

msgsnd — XSI message send operation

SYNOPSIS

XSI
```
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

DESCRIPTION

The *msgsnd*( ) function operates on XSI message queues (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *msgsnd*( ) function shall send a message to the queue associated with the message queue identifier specified by *msqid*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long   mtype;       /* Message type. */
    char   mtext[1];    /* Message text. */
}
```

The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving process for message selection.

The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range from 0 to a system-imposed maximum.

The argument *msgflg* specifies the action to be taken if one or more of the following is true:

- The number of bytes already on the queue is equal to **msg_qbytes**; see **<sys/msg.h>**.
- The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

- If (*msgflg* & IPC_NOWAIT) is non-zero, the message shall not be sent and the calling thread shall return immediately.
- If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:
  - — The condition responsible for the suspension no longer exists, in which case the message is sent.
  - — The message queue identifier *msqid* is removed from the system; when this occurs, *errno* shall be set equal to [EIDRM] and –1 shall be returned.
  - — The calling thread receives a signal that is to be caught; in this case the message is not sent and the calling thread resumes execution in the manner prescribed in *sigaction*( ).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*; see **<sys/msg.h>**:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- **msg_qnum** shall be incremented by 1.
- **msg_lspid** shall be set equal to the process ID of the calling process.
- **msg_stime** shall be set equal to the current time.

RETURN VALUE

Upon successful completion, *msgsnd*( ) shall return 0; otherwise, no message shall be sent, *msgsnd*( ) shall return –1, and *errno* shall be set to indicate the error.

ERRORS

The *msgsnd*( ) function shall fail if:

[EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page 39).

[EAGAIN] The message cannot be sent for one of the reasons cited above and (*msgflg* & IPC_NOWAIT) is non-zero.

[EIDRM] The message queue identifier *msqid* is removed from the system.

[EINTR] The *msgsnd*( ) function was interrupted by a signal.

[EINVAL] The value of *msqid* is not a valid message queue identifier, or the value of *mtype* is less than 1; or the value of *msgsz* is less than 0 or greater than the system-imposed limit.

EXAMPLES

Sending a Message

The following example sends a message to the queue identified by the *msqid* argument (assuming that value has previously been set). This call specifies that an error should be reported if no message is available. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
    long type;
    char text[20];
} msg;

msg.type = 1;
strcpy(msg.text, "This is message 1");
...
result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

26618 **RATIONALE**

26619 None.

26620 **FUTURE DIRECTIONS**

26621 None.

26622 **SEE ALSO**

26623 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ),
26624 *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ),
26625 *sigaction*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/msg.h>**

26626 **CHANGE HISTORY**

26627 First released in Issue 2. Derived from Issue 2 of the SVID.

26628 **Issue 5**

26629 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
26630 DIRECTIONS to a new APPLICATION USAGE section.

26631 **Issue 6**

26632 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

msync — synchronize memory with physical storage

SYNOPSIS

MF SIO

```
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);
```

DESCRIPTION

The *msync*( ) function shall write all modified data to permanent storage locations, if any, in those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes. If no such storage exists, *msync*( ) need not have any effect. If requested, the *msync*( ) function shall then invalidate cached copies of data.

The implementation shall require that *addr* be a multiple of the page size as returned by *sysconf*( ).

For mappings to files, the *msync*( ) function shall ensure that all write operations are completed as defined for synchronized I/O data integrity completion. It is unspecified whether the implementation also writes out other file attributes. When the *msync*( ) function is called on MAP_PRIVATE mappings, any modified data shall not be written to the underlying object and shall not cause such data to be made visible to other processes. It is unspecified whether data in

SHM|TYM MAP_PRIVATE mappings has any permanent storage locations. The effect of *msync*( ) on a shared memory object or a typed memory object is unspecified. The behavior of this function is unspecified if the mapping was not established by a call to *mmap*( ).

The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following flags defined in the **<sys/mman.h>** header:

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data.

When MS_ASYNC is specified, *msync*( ) shall return immediately once all the write operations are initiated or queued for servicing; when MS_SYNC is specified, *msync*( ) shall not return until all write operations are completed as defined for synchronized I/O data integrity completion. Either MS_ASYNC or MS_SYNC is specified, but not both.

When MS_INVALIDATE is specified, *msync*( ) shall invalidate all cached copies of mapped data that are inconsistent with the permanent storage locations such that subsequent references shall obtain data that was consistent with the permanent storage locations sometime between the call to *msync*( ) and the first subsequent memory reference to the data.

If *msync*( ) causes any write to a file, the file's *st_ctime* and *st_mtime* fields shall be marked for update.

RETURN VALUE

Upon successful completion, *msync*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *msync*( ) function shall fail if:

[EBUSY] Some or all of the addresses in the range starting at *addr* and continuing for *len* bytes are locked, and MS_INVALIDATE is specified.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[EINVAL] The value of *flags* is invalid.

[EINVAL] The value of *addr* is not a multiple of the page size {PAGESIZE}.

[ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process or specify one or more pages that are not mapped.

EXAMPLES

None.

APPLICATION USAGE

The *msync*( ) function is only supported if the Memory Mapped Files option and the Synchronized Input and Output option are supported, and thus need not be available on all implementations.

The *msync*( ) function should be used by programs that require a memory object to be in a known state; for example, in building transaction facilities.

Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that *msync*( ) is the only control over when pages are or are not written to disk.

RATIONALE

The *msync*( ) function writes out data in a mapped region to the permanent storage for the underlying object. The call to *msync*( ) ensures data integrity of the file.

After the data is written out, any cached data may be invalidated if the MS_INVALIDATE flag was specified. This is useful on systems that do not support read/write consistency.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *msync*( ) in the POSIX Realtime Extension as follows:

- The DESCRIPTION is extensively reworded.
- [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

Issue 6

The *msync*( ) function is marked as part of the Memory Mapped Files and Synchronized Input and Output options.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The [EBUSY] mandatory error condition is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
- The second [EINVAL] error condition is made mandatory.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to typed memory objects.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

26721 **NAME**

26722 munlock — unlock a range of process address space

26723 **SYNOPSIS**

26724 MLR
```
#include <sys/mman.h>
```

26725
```
int munlock(const void *addr, size_t len);
```

26726

26727 **DESCRIPTION**

26728 Refer to *mlock*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

munlockall — unlock the address space of a process

SYNOPSIS

ML

```
#include <sys/mman.h>

int munlockall(void);
```

DESCRIPTION

Refer to *mlockall*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

munmap — unmap pages of memory

SYNOPSIS

MC3

```
#include <sys/mman.h>

int munmap(void *addr, size_t len);
```

DESCRIPTION

The *munmap*( ) function shall remove any mappings for those entire pages containing any part of the address space of the process starting at *addr* and continuing for *len* bytes. Further references to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no mappings in the specified address range, then *munmap*( ) has no effect.

The implementation shall require that *addr* be a multiple of the page size {PAGESIZE}.

If a mapping to be removed was private, any modifications made in this address range shall be discarded.

ML | MLR

Any memory locks (see *mlock*( ) and *mlockall*( )) associated with this address range shall be removed, as if by an appropriate call to *munlock*( ).

TYM

If a mapping removed from a typed memory object causes the corresponding address range of the memory pool to be inaccessible by any process in the system except through allocatable mappings (that is, mappings of typed memory objects opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag), then that range of the memory pool shall become deallocated and may become available to satisfy future typed memory allocation requests.

A mapping removed from a typed memory object opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag shall not affect in any way the availability of that typed memory for allocation.

The behavior of this function is unspecified if the mapping was not established by a call to *mmap*( ).

RETURN VALUE

Upon successful completion, *munmap*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *munmap*( ) function shall fail if:

[EINVAL] Addresses in the range [*addr*,*addr*+*len*) are outside the valid range for the address space of a process.

[EINVAL] The *len* argument is 0.

[EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The *munmap*( ) function is only supported if the Memory Mapped Files option or the Shared Memory Objects option is supported.

RATIONALE

The *munmap*( ) function corresponds to SVR4, just as the *mmap*( ) function does.

It is possible that an application has applied process memory locking to a region that contains shared memory. If this has occurred, the *munmap*( ) call ignores those locks and, if necessary, causes those locks to be removed.

FUTURE DIRECTIONS

None.

SEE ALSO

mlock( ), *mlockall*( ), *mmap*( ), *posix_typed_mem_open*( ), *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *munmap*( ) in the POSIX Realtime Extension as follows:

- The DESCRIPTION is extensively reworded.
- The SIGBUS error is no longer permitted to be generated.

Issue 6

The *munmap*( ) function is marked as part of the Memory Mapped Files and Shared Memory Objects option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
- The [EINVAL] error conditions are added.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- Semantics for typed memory objects are added to the DESCRIPTION.
- The *posix_typed_mem_open*( ) function is added to the SEE ALSO section.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

nan, nanf, nanl — return quiet NaN

SYNOPSIS

```
#include <math.h>

double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The function call *nan*("*n-char-sequence*") shall be equivalent to:

```
strtod("NAN(n-char-sequence)", (char **) NULL);
```

The function call *nan*("") shall be equivalent to:

```
strtod("NAN()", (char **) NULL)
```

If *tagp* does not point to an *n*-**char** sequence or an empty string, the function call shall be equivalent to:

```
strtod("NAN", (char **) NULL)
```

Function calls to *nanf*( ) and *nanl*( ) are equivalent to the corresponding function calls to *strtof*( ) and *strtold*( ).

RETURN VALUE

These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

If the implementation does not support quiet NaNs, these functions shall return zero.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtod( ), *strtold*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

nanosleep — high resolution sleep (**REALTIME**)

SYNOPSIS

TMR
```
#include <time.h>

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

DESCRIPTION

The *nanosleep*( ) function shall cause the current thread to be suspended from execution until either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the calling thread, and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time shall not be less than the time specified by *rqtp*, as measured by the system clock CLOCK_REALTIME.

The use of the *nanosleep*( ) function has no effect on the action or blockage of any signal.

RETURN VALUE

If the *nanosleep*( ) function returns because the requested time has elapsed, its return value shall be zero.

If the *nanosleep*( ) function returns because it has been interrupted by a signal, it shall return a value of −1 and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the **timespec** structure referenced by it is updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). If the *rmtp* argument is NULL, the remaining time is not returned.

If *nanosleep*( ) fails, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *nanosleep*( ) function shall fail if:

[EINTR] The *nanosleep*( ) function was interrupted by a signal.

[EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

It is common to suspend execution of a process for an interval in order to poll the status of a non-interrupting function. A large number of actual needs can be met with a simple extension to *sleep*( ) that provides finer resolution.

In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the frequency of wakeup is limited by the resolution of the *alarm*( ) and *sleep*( ) functions. In 4.3 BSD, it is possible to write such a routine using no static storage and reserving no system facilities. Although it is possible to write a function with similar functionality to *sleep*( ) using the remainder of the *timer_**( ) functions, such a function requires the use of signals and the reservation of some signal number. This volume of IEEE Std 1003.1-2001 requires that *nanosleep*( ) be non-intrusive of the signals function.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *nanosleep*( ) function shall return a value of 0 on success and –1 on failure or if interrupted. This latter case is different from *sleep*( ). This was done because the remaining time is returned via an argument structure pointer, *rmtp*, instead of as the return value.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_nanosleep( ), *sleep*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *nanosleep*( ) function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO section to include the *clock_nanosleep*( ) function.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

SYNOPSIS

```
#include <math.h>

double nearbyint(double x);
float nearbyintf(float x);
long double nearbyintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall round their argument to an integer value in floating-point format, using the current rounding direction and without raising the inexact floating-point exception.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, ±0 shall be returned.

If *x* is ±Inf, *x* shall be returned.

XSI If the correct value would cause overflow, a range error shall occur and *nearbyint*( ), *nearbyintf*( ), and *nearbyintl*( ) shall return the value of the macro ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*), respectively.

ERRORS

These functions shall fail if:

XSI

Range Error	The result would cause an overflow.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE ALSO

feclearexcept( ), *fetestexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable floating-point number

SYNOPSIS

```
#include <math.h>

double nextafter(double x, double y);
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *nextafter*( ), *nextafterf*( ), and *nextafterl*( ) functions shall compute the next representable floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter*( ) shall return the largest representable floating-point number less than *x*. The *nextafter*( ), *nextafterf*( ), and *nextafterl*( ) functions shall return *y* if *x* equals *y*.

The *nexttoward*( ), *nexttowardf*( ), and *nexttowardl*( ) functions shall be equivalent to the corresponding *nextafter*( ) functions, except that the second parameter shall have type **long double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the next representable floating-point value following *x* in the direction of *y*.

If *x*==*y*, *y* (of the type *x*) shall be returned.

If *x* is finite and the correct function value would overflow, a range error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for the return type of the function.

MX If *x* or *y* is NaN, a NaN shall be returned.

If *x*!=*y* and the correct function value is subnormal, zero, or underflows, a range error shall occur, and either the correct function value (if representable) or 0.0 shall be returned.

ERRORS

These functions shall fail if:

Range Error	The correct value overflows.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.
MX Range Error	The correct value is subnormal or underflows.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *nextafter*( ) function is no longer marked as an extension.

The *nextafterf*( ), *nextafterl*( ), *nexttoward*( ), *nexttowardf*( ), and *nexttowardl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

nftw — walk a file tree

SYNOPSIS

XSI

```
#include <ftw.h>

int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int depth, int flags);
```

DESCRIPTION

The *nftw*( ) function shall recursively descend the directory hierarchy rooted in *path*. The *nftw*( ) function has a similar effect to *ftw*( ) except that it takes an additional argument *flags*, which is a bitwise-inclusive OR of zero or more of the following flags:

FTW_CHDIR — If set, *nftw*( ) shall change the current working directory to each directory as it reports files in that directory. If clear, *nftw*( ) shall not change the current working directory.

FTW_DEPTH — If set, *nftw*( ) shall report all files in a directory before reporting the directory itself. If clear, *nftw*( ) shall report any directory before reporting the files in that directory.

FTW_MOUNT — If set, *nftw*( ) shall only report files in the same file system as *path*. If clear, *nftw*( ) shall report all files encountered during the walk.

FTW_PHYS — If set, *nftw*( ) shall perform a physical walk and shall not follow symbolic links.

If FTW_PHYS is clear and FTW_DEPTH is set, *nftw*( ) shall follow links instead of reporting them, but shall not report any directory that would be a descendant of itself. If FTW_PHYS is clear and FTW_DEPTH is clear, *nftw*( ) shall follow links instead of reporting them, but shall not report the contents of any directory that would be a descendant of itself.

At each file it encounters, *nftw*( ) shall call the user-supplied function *fn* with four arguments:

- The first argument is the pathname of the object.
- The second argument is a pointer to the **stat** buffer containing information on the object.
- The third argument is an integer giving additional information. Its value is one of the following:

  FTW_F — The object is a file.

  FTW_D — The object is a directory.

  FTW_DP — The object is a directory and subdirectories have been visited. (This condition shall only occur if the FTW_DEPTH flag is included in *flags*.)

  FTW_SL — The object is a symbolic link. (This condition shall only occur if the FTW_PHYS flag is included in *flags*.)

  FTW_SLN — The object is a symbolic link that does not name an existing file. (This condition shall only occur if the FTW_PHYS flag is not included in *flags*.)

  FTW_DNR — The object is a directory that cannot be read. The *fn* function shall not be called for any of its descendants.

  FTW_NS — The *stat*( ) function failed on the object because of lack of appropriate permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat*( ) for any other reason is considered an error and *nftw*( ) shall return –1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the object's filename in the pathname passed as the first argument to *fn*. The value of **level** indicates depth relative to the root of the walk, where the root level is 0.

The results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

The argument *depth* sets the maximum number of file descriptors that shall be used by *nftw*( ) while traversing the file tree. At most one file descriptor shall be used for each directory level.

The *nftw*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *nftw*( ) function shall continue until the first of the following conditions occurs:

- An invocation of *fn* shall return a non-zero value, in which case *nftw*( ) shall return that value.
- The *nftw*( ) function detects an error other than [EACCES] (see FTW_DNR and FTW_NS above), in which case *nftw*( ) shall return –1 and set *errno* to indicate the error.
- The tree is exhausted, in which case *nftw*( ) shall return 0.

ERRORS

The *nftw*( ) function shall fail if:

[EACCES] Search permission is denied for any component of *path* or read permission is denied for *path*, or *fn* returns –1 and does not reset *errno*.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of *path* is not a directory.

[EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current programming environment for one or more files found in the file hierarchy.

The *nftw*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

The following example walks the **/tmp** directory and its subdirectories, calling the *nftw*() function for every directory entry, to a maximum of 5 levels deep.

```
#include <ftw.h>
...
int nftwfunc(const char *, const struct stat *, int, struct FTW *);

int nftwfunc(const char *filename, const struct stat *statptr,
    int fileflags, struct FTW *pfwt)
{
    return 0;
}
...
char *startpath = "/tmp";
int depth = 5;
int flags = FTW_CHDIR | FTW_DEPTH | FTW_MOUNT;
int ret;

ret = nftw(startpath, nftwfunc, depth, flags);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

lstat( ), *opendir*( ), *readdir*( ), *stat*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<ftw.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the DESCRIPTION, the definition of the *depth* argument is clarified.

Issue 6

The Open Group Base Resolution bwg97-003 is applied.

The ERRORS section is updated as follows:

- The wording of the mandatory [ELOOP] error condition is updated.
- A second optional [ELOOP] error condition is added.
- The [EOVERFLOW] mandatory error condition is added.

Text is added to the DESCRIPTION to say that the *nftw*() function need not be reentrant and that the results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

nice — change the nice value of a process

SYNOPSIS

XSI
```
#include <unistd.h>

int nice(int incr);
```

DESCRIPTION

The *nice*( ) function shall add the value of *incr* to the nice value of the calling process. A process' nice value is a non-negative number for which a more positive value shall result in less favorable scheduling.

A maximum nice value of 2*{NZERO}–1 and a minimum nice value of 0 shall be imposed by the system. Requests for values above or below these limits shall result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

PS | TPS Calling the *nice*( ) function has no effect on the priority of processes or threads with policy SCHED_FIFO or SCHED_RR. The effect on processes or threads with other scheduling policies is implementation-defined.

The nice value set with *nice*( ) shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

As –1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *nice*( ), and if it returns –1, check to see whether *errno* is non-zero.

RETURN VALUE

Upon successful completion, *nice*( ) shall return the new nice value –{NZERO}. Otherwise, –1 shall be returned, the process' nice value shall not be changed, and *errno* shall be set to indicate the error.

ERRORS

The *nice*( ) function shall fail if:

[EPERM] The *incr* argument is negative and the calling process does not have appropriate privileges.

EXAMPLES

Changing the Nice Value

The following example adds the value of the *incr* argument, –20, to the nice value of the calling process.

```
#include <unistd.h>
...
int incr = -20;
int ret;

ret = nice(incr);
```

APPLICATION USAGE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27180 **RATIONALE**

27181 None.

27182 **FUTURE DIRECTIONS**

27183 None.

27184 **SEE ALSO**

27185 *getpriority*( ), *setpriority*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**,
27186 **<unistd.h>**

27187 **CHANGE HISTORY**

27188 First released in Issue 1. Derived from Issue 1 of the SVID.

27189 **Issue 5**

27190 A statement is added to the description indicating the effects of this function on the different
27191 scheduling policies and multi-threaded processes.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

27192 **NAME**

27193 nl_langinfo — language information

27194 **SYNOPSIS**

27195 XSI
```
#include <langinfo.h>
```

27196
```
char *nl_langinfo(nl_item item);
```

27197

27198 **DESCRIPTION**

27199 The *nl_langinfo*( ) function shall return a pointer to a string containing information relevant to
27200 the particular language or cultural area defined in the program's locale (see **<langinfo.h>**). The
27201 manifest constant names and values of *item* are defined in **<langinfo.h>**. For example:

27202
```
nl_langinfo(ABDAY_1)
```

27203 would return a pointer to the string `"Dom"` if the identified language was Portuguese, and
27204 `"Sun"` if the identified language was English.

27205 Calls to *setlocale*( ) with a category corresponding to the category of *item* (see **<langinfo.h>**), or to
27206 the category *LC_ALL*, may overwrite the array pointed to by the return value.

27207 The *nl_langinfo*( ) function need not be reentrant. A function that is not required to be reentrant is
27208 not required to be thread-safe.

27209 **RETURN VALUE**

27210 In a locale where *langinfo* data is not defined, *nl_langinfo*( ) shall return a pointer to the
27211 corresponding string in the POSIX locale. In all locales, *nl_langinfo*( ) shall return a pointer to an
27212 empty string if *item* contains an invalid setting.

27213 This pointer may point to static data that may be overwritten on the next call.

27214 **ERRORS**

27215 No errors are defined.

27216 **EXAMPLES**

27217 **Getting Date and Time Formatting Information**

27218 The following example returns a pointer to a string containing date and time formatting
27219 information, as defined in the *LC_TIME* category of the current locale.

27220
```
#include <time.h>
```
27221
```
#include <langinfo.h>
```
27222
```
...
```
27223
```
strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
```
27224
```
...
```

27225 **APPLICATION USAGE**

27226 The array pointed to by the return value should not be modified by the program, but may be
27227 modified by further calls to *nl_langinfo*( ).

27228 **RATIONALE**

27229 None.

27230 **FUTURE DIRECTIONS**

27231 None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27232 **SEE ALSO**

27233 *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<langinfo.h>**,
27234 **<nl_types.h>**

27235 **CHANGE HISTORY**

27236 First released in Issue 2.

27237 **Issue 5**

27238 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

27239 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

nrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS

XSI

```
#include <stdlib.h>

long nrand48(unsigned short xsubi[3]);
```

DESCRIPTION

Refer to *drand48*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27248 **NAME**

27249 ntohl, ntohs — convert values between host and network byte order

27250 **SYNOPSIS**

27251 `#include <arpa/inet.h>`

27252 `uint32_t ntohl(uint32_t` *`netlong`*`);`

27253 `uint16_t ntohs(uint16_t` *`netshort`*`);`

27254 **DESCRIPTION**

27255 Refer to *htonl*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

open — open a file

SYNOPSIS

OH
```
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ... );
```

DESCRIPTION

The *open*( ) function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The *path* argument points to a pathname naming the file.

The *open*( ) function shall return a file descriptor for the named file that is the lowest file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared.

The file offset used to mark the current position within the file shall be set to the beginning of the file.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**. Applications shall specify exactly one of the first three values (file access modes) below in the value of *oflag*:

O_RDONLY Open for reading only.

O_WRONLY Open for writing only.

O_RDWR Open for reading and writing. The result is undefined if this flag is applied to a FIFO.

Any combination of the following may be used:

O_APPEND If set, the file offset shall be set to the end of the file prior to each write.

O_CREAT If the file exists, this flag has no effect except as noted under O_EXCL below. Otherwise, the file shall be created; the user ID of the file shall be set to the effective user ID of the process; the group ID of the file shall be set to the group ID of the file's parent directory or to the effective group ID of the process; and the access permission bits (see **<sys/stat.h>**) of the file mode shall be set to the value of the third argument taken as type **mode_t** modified as follows: a bitwise AND is performed on the file-mode bits and the corresponding bits in the complement of the process' file mode creation mask. Thus, all bits in the file mode whose corresponding bit in the file mode creation mask is set are cleared. When bits other than the file permission bits are set, the effect is unspecified. The third argument does not affect whether the file is open for reading, writing, or for both. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process.

SIO O_DSYNC Write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

O_EXCL — If O_CREAT and O_EXCL are set, *open*( ) shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing *open*( ) naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and *path* names a symbolic link, *open*( ) shall fail and set *errno* to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_NOCTTY — If set and *path* identifies a terminal device, *open*( ) shall not cause the terminal device to become the controlling terminal for the process.

O_NONBLOCK — When opening a FIFO with O_RDONLY or O_WRONLY set:

- If O_NONBLOCK is set, an *open*( ) for reading-only shall return without delay. An *open*( ) for writing-only shall return an error if no process currently has the file open for reading.
- If O_NONBLOCK is clear, an *open*( ) for reading-only shall block the calling thread until a thread opens the file for writing. An *open*( ) for writing-only shall block the calling thread until a thread opens the file for reading.

When opening a block special or character special file that supports non-blocking opens:

- If O_NONBLOCK is set, the *open*( ) function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.
- If O_NONBLOCK is clear, the *open*( ) function shall block the calling thread until the device is ready or available before returning.

Otherwise, the behavior of O_NONBLOCK is unspecified.

SIO O_RSYNC — Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in *oflag*, all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

SIO O_SYNC — Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

O_TRUNC — If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC with O_RDONLY is undefined.

If O_CREAT is set and the file did not previously exist, upon successful completion, *open*( ) shall mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file and the *st_ctime* and *st_mtime* fields of the parent directory.

If O_TRUNC is set and the file did previously exist, upon successful completion, *open*( ) shall mark for update the *st_ctime* and *st_mtime* fields of the file.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SIO If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.

XSR If *path* refers to a STREAMS file, *oflag* may be constructed from O_NONBLOCK OR'ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS devices and shall have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.

XSI If *path* names the master side of a pseudo-terminal device, then it is unspecified whether *open*( ) locks the slave side so that it cannot be opened. Conforming applications shall call *unlockpt*( ) before opening the slave side.

The largest value that can be represented correctly in an object of type **off_t** shall be established as the offset maximum in the open file description.

RETURN VALUE

Upon successful completion, the function shall open the file and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, –1 shall be returned and *errno* set to indicate the error. No files shall be created or modified if the function returns –1.

ERRORS

The *open*( ) function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *oflag* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or O_TRUNC is specified and write permission is denied.

[EEXIST] O_CREAT and O_EXCL are set, and the named file exists.

[EINTR] A signal was caught during *open*( ).

SIO [EINVAL] The implementation does not support synchronized I/O for this file.

XSR [EIO] The *path* argument names a STREAMS file and a hangup or error occurred during the *open*( ).

[EISDIR] The named file is a directory and *oflag* includes O_WRONLY or O_RDWR.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENOENT] O_CREAT is not set and the named file does not exist; or O_CREAT is set and either the path prefix does not exist or the *path* argument points to an empty string.

XSR [ENOSR] The *path* argument names a STREAMS-based file and the system is unable to allocate a STREAM.

[ENOSPC] The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[ENOTDIR] A component of the path prefix is not a directory.

[ENXIO] O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.

[ENXIO] The named file is a character special or block special file, and the device associated with this special file does not exist.

[EOVERFLOW] The named file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

[EROFS] The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the *oflag* argument.

The *open*( ) function may fail if:

XSI [EAGAIN] The *path* argument names the slave side of a pseudo-terminal device that is locked.

[EINVAL] The value of the *oflag* argument is not valid.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

XSR [ENOMEM] The *path* argument names a STREAMS file and the system is unable to allocate resources.

[ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *oflag* is O_WRONLY or O_RDWR.

EXAMPLES

Opening a File for Writing by the Owner

The following example opens the file **/tmp/file**, either by creating it (if it does not already exist), or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file, the access permission bits in the file mode of the file are set to permit reading and writing by the owner, and to permit reading only by group members and others.

If the call to *open*( ) is successful, the file is opened for writing.

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *filename = "/tmp/file";
...
fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
...
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27426 **Opening a File Using an Existence Check**

27427 The following example uses the *open*( ) function to try to create the **LOCKFILE** file and open it
27428 for writing. Since the *open*( ) function specifies the O_EXCL flag, the call fails if the file already
27429 exists. In that case, the program assumes that someone else is updating the password file and
27430 exits.

```
27431 #include <fcntl.h>
27432 #include <stdio.h>
27433 #include <stdlib.h>

27434 #define LOCKFILE "/etc/ptmp"
27435 ...
27436 int pfd; /* Integer for file descriptor returned by open() call. */
27437 ...
27438 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
27439     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27440 {
27441     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
27442     exit(1);
27443 }
27444 ...
```

27445 **Opening a File for Writing**

27446 The following example opens a file for writing, creating the file if it does not already exist. If the
27447 file does exist, the system truncates the file to zero bytes.

```
27448 #include <fcntl.h>
27449 #include <stdio.h>
27450 #include <stdlib.h>

27451 #define LOCKFILE "/etc/ptmp"
27452 ...
27453 int pfd;
27454 char filename[PATH_MAX+1];
27455 ...
27456 if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
27457     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27458 {
27459     perror("Cannot open output file\n"); exit(1);
27460 }
27461 ...
```

27462 **APPLICATION USAGE**

27463 None.

27464 **RATIONALE**

27465 Except as specified in this volume of IEEE Std 1003.1-2001, the flags allowed in *oflag* are not
27466 mutually-exclusive and any number of them may be used simultaneously.

27467 Some implementations permit opening FIFOs with O_RDWR. Since FIFOs could be
27468 implemented in other ways, and since two file descriptors can be used to the same effect, this
27469 possibility is left as undefined.

27470 See *getgroups*( ) about the group of a newly created file.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The use of *open*( ) to create a regular file is preferable to the use of *creat*( ), because the latter is redundant and included only for historical reasons.

The use of the O_TRUNC flag on FIFOs and directories (pipes cannot be *open*( )-ed) must be permissible without unexpected side effects (for example, *creat*( ) on a FIFO must not remove data). Since terminal special files might have type-ahead data stored in the buffer, O_TRUNC should not affect their content, particularly if a program that normally opens a regular file should open the current controlling terminal instead. Other file types, particularly implementation-defined ones, are left implementation-defined.

IEEE Std 1003.1-2001 permits [EACCES] to be returned for conditions other than those explicitly listed.

The O_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a controlling terminal as a side effect of opening a terminal file. This volume of IEEE Std 1003.1-2001 does not specify how a controlling terminal is acquired, but it allows an implementation to provide this on *open*( ) if the O_NOCTTY flag is not set and other conditions specified in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface are met. The O_NOCTTY flag is an effective no-op if the file being opened is not a terminal device.

In historical implementations the value of O_RDONLY is zero. Because of that, it is not possible to detect the presence of O_RDONLY and another option. Future implementations should encode O_RDONLY and O_WRONLY as bit flags so that:

```
O_RDONLY | O_WRONLY == O_RDWR
```

In general, the *open*( ) function follows the symbolic link if *path* names a symbolic link. However, the *open*( ) function, when called with O_CREAT and O_EXCL, is required to fail with [EEXIST] if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This behavior is required so that privileged applications can create a new file in a known location without the possibility that a symbolic link might cause the file to be created in a different location.

For example, a privileged application that must create a file with a predictable name in a user-writable directory, such as the user's home directory, could be compromised if the user creates a symbolic link with that name that refers to a nonexistent file in a system directory. If the user can influence the contents of a file, the user could compromise the system by creating a new system configuration or spool file that would then be interpreted by the system. The test for a symbolic link which refers to a nonexisting file must be atomic with the creation of a new file.

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*( ) to set the group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *close*( ), *creat*( ), *dup*( ), *fcntl*( ), *lseek*( ), *read*( ), *umask*( ), *unlockpt*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<sys/stat.h>**, **<sys/types.h>**

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27517 **CHANGE HISTORY**

27518 First released in Issue 1. Derived from Issue 1 of the SVID.

27519 **Issue 5**

27520 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
27521 Threads Extension.

27522 Large File Summit extensions are added.

27523 **Issue 6**

27524 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

27525 The following new requirements on POSIX implementations derive from alignment with the
27526 Single UNIX Specification:

27527 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
27528 required for conforming implementations of previous POSIX specifications, it was not
27529 required for UNIX applications.

27530 • In the DESCRIPTION, O_CREAT is amended to state that the group ID of the file is set to the
27531 group ID of the file's parent directory or to the effective group ID of the process. This is a
27532 FIPS requirement.

27533 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
27534 description. This change is to support large files.

27535 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
27536 large files.

27537 • The [ENXIO] mandatory error condition is added.

27538 • The [EINVAL], [ENAMETOOLONG], and [ETXTBSY] optional error conditions are added.

27539 The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI
27540 STREAMS Option Group are marked.

27541 The following changes were made to align with the IEEE P1003.1a draft standard:

27542 • An explanation is added of the effect of the O_CREAT and O_EXCL flags when the path
27543 refers to a symbolic link.

27544 • The [ELOOP] optional error condition is added.

27545 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

27546 The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27547 **NAME**

27548 opendir — open a directory

27549 **SYNOPSIS**

27550 `#include <dirent.h>`

27551 `DIR *opendir(const char *`*`dirname`*`);`

27552 **DESCRIPTION**

27553 The *opendir*( ) function shall open a directory stream corresponding to the directory named by
27554 the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is
27555 implemented using a file descriptor, applications shall only be able to open up to a total of
27556 {OPEN_MAX} files and directories.

27557 **RETURN VALUE**

27558 Upon successful completion, *opendir*( ) shall return a pointer to an object of type **DIR**.
27559 Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

27560 **ERRORS**

27561 The *opendir*( ) function shall fail if:

27562 [EACCES] Search permission is denied for the component of the path prefix of *dirname* or
27563 read permission is denied for *dirname*.

27564 [ELOOP] A loop exists in symbolic links encountered during resolution of the *dirname*
27565 argument.

27566 [ENAMETOOLONG]
27567 The length of the *dirname* argument exceeds {PATH_MAX} or a pathname
27568 component is longer than {NAME_MAX}.

27569 [ENOENT] A component of *dirname* does not name an existing directory or *dirname* is an
27570 empty string.

27571 [ENOTDIR] A component of *dirname* is not a directory.

27572 The *opendir*( ) function may fail if:

27573 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
27574 resolution of the *dirname* argument.

27575 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

27576 [ENAMETOOLONG]
27577 As a result of encountering a symbolic link in resolution of the *dirname*
27578 argument, the length of the substituted pathname string exceeded
27579 {PATH_MAX}.

27580 [ENFILE] Too many files are currently open in the system.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27581 **EXAMPLES**

27582 **Open a Directory Stream**

27583 The following program fragment demonstrates how the *opendir*( ) function is used.

```
27584 #include <sys/types.h>
27585 #include <dirent.h>
27586 #include <libgen.h>
27587 ...
27588     DIR *dir;
27589     struct dirent *dp;
27590 ...
27591     if ((dir = opendir (".")) == NULL) {
27592         perror ("Cannot open .");
27593         exit (1);
27594     }

27595     while ((dp = readdir (dir)) != NULL) {
27596 ...
```

27597 **APPLICATION USAGE**

27598 The *opendir*( ) function should be used in conjunction with *readdir*( ), *closedir*( ), and *rewinddir*( ) to
27599 examine the contents of the directory (see the EXAMPLES section in *readdir*( )). This method is
27600 recommended for portability.

27601 **RATIONALE**

27602 Based on historical implementations, the rules about file descriptors apply to directory streams
27603 as well. However, this volume of IEEE Std 1003.1-2001 does not mandate that the directory
27604 stream be implemented using file descriptors. The description of *closedir*( ) clarifies that if a file
27605 descriptor is used for the directory stream, it is mandatory that *closedir*( ) deallocate the file
27606 descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the
27607 FD_CLOEXEC had been set for the file descriptor.

27608 The directory entries for dot and dot-dot are optional. This volume of IEEE Std 1003.1-2001 does
27609 not provide a way to test *a priori* for their existence because an application that is portable must
27610 be written to look for (and usually ignore) those entries. Writing code that presumes that they
27611 are the first two entries does not always work, as many implementations permit them to be
27612 other than the first two entries, with a ‘‘normal’’ entry preceding them. There is negligible value
27613 in providing a way to determine what the implementation does because the code to deal with
27614 dot and dot-dot must be written in any case and because such a flag would add to the list of
27615 those flags (which has proven in itself to be objectionable) and might be abused.

27616 Since the structure and buffer allocation, if any, for directory operations are defined by the
27617 implementation, this volume of IEEE Std 1003.1-2001 imposes no portability requirements for
27618 erroneous program constructs, erroneous data, or the use of unspecified values such as the use
27619 or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed
27620 or after a *fork*( ) or one of the *exec* function calls.

27621 **FUTURE DIRECTIONS**

27622 None.

27623 **SEE ALSO**

27624 *closedir*( ), *lstat*( ), *readdir*( ), *rewinddir*( ), *symlink*( ), the Base Definitions volume of
27625 IEEE Std 1003.1-2001, **<dirent.h>**, **<limits.h>**, **<sys/types.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27626 **CHANGE HISTORY**

27627 First released in Issue 2.

27628 **Issue 6**

27629 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

27630 The following new requirements on POSIX implementations derive from alignment with the
27631 Single UNIX Specification:

27632 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
27633 required for conforming implementations of previous POSIX specifications, it was not
27634 required for UNIX applications.

27635 • The [ELOOP] mandatory error condition is added.

27636 • A second [ENAMETOOLONG] is added as an optional error condition.

27637 The following changes were made to align with the IEEE P1003.1a draft standard:

27638 • The [ELOOP] optional error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27639 **NAME**

27640 openlog — open a connection to the logging facility

27641 **SYNOPSIS**

27642 XSI
```
#include <syslog.h>

void openlog(const char *ident, int logopt, int facility);
```
27643

27644

27645 **DESCRIPTION**

27646 Refer to *closelog*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27647 **NAME**

27648 optarg, opterr, optind, optopt — options parsing variables

27649 **SYNOPSIS**

27650
```
#include <unistd.h>
```

27651
```
extern char *optarg;
```
27652
```
extern int opterr, optind, optopt;
```

27653 **DESCRIPTION**

27654 Refer to *getopt*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27655 **NAME**

27656 pathconf — get configurable pathname variables

27657 **SYNOPSIS**

27658 `#include <unistd.h>`

27659 `long pathconf(const char *`*`path`*`, int `*`name`*`);`

27660 **DESCRIPTION**

27661 Refer to *fpathconf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27662 **NAME**

27663 pause — suspend the thread until a signal is received

27664 **SYNOPSIS**

27665
```
#include <unistd.h>
```

27666
```
int pause(void);
```

27667 **DESCRIPTION**

27668 The *pause*( ) function shall suspend the calling thread until delivery of a signal whose action is
27669 either to execute a signal-catching function or to terminate the process.

27670 If the action is to terminate the process, *pause*( ) shall not return.

27671 If the action is to execute a signal-catching function, *pause*( ) shall return after the signal-catching
27672 function returns.

27673 **RETURN VALUE**

27674 Since *pause*( ) suspends thread execution indefinitely unless interrupted by a signal, there is no
27675 successful completion return value. A value of –1 shall be returned and *errno* set to indicate the
27676 error.

27677 **ERRORS**

27678 The *pause*( ) function shall fail if:

27679 [EINTR] A signal is caught by the calling process and control is returned from the
27680 signal-catching function.

27681 **EXAMPLES**

27682 None.

27683 **APPLICATION USAGE**

27684 Many common uses of *pause*( ) have timing windows. The scenario involves checking a
27685 condition related to a signal and, if the signal has not occurred, calling *pause*( ). When the signal
27686 occurs between the check and the call to *pause*( ), the process often blocks indefinitely. The
27687 *sigprocmask*( ) and *sigsuspend*( ) functions can be used to avoid this type of problem.

27688 **RATIONALE**

27689 None.

27690 **FUTURE DIRECTIONS**

27691 None.

27692 **SEE ALSO**

27693 *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

27694 **CHANGE HISTORY**

27695 First released in Issue 1. Derived from Issue 1 of the SVID.

27696 **Issue 5**

27697 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

27698 **Issue 6**

27699 The APPLICATION USAGE section is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pclose — close a pipe stream to or from a process

SYNOPSIS

CX
```
#include <stdio.h>

int pclose(FILE *stream);
```

DESCRIPTION

The *pclose*( ) function shall close a stream that was opened by *popen*( ), wait for the command to terminate, and return the termination status of the process that was running the command language interpreter. However, if a call caused the termination status to be unavailable to *pclose*( ), then *pclose*( ) shall return –1 with *errno* set to [ECHILD] to report this situation. This can happen if the application calls one of the following functions:

- *wait*( )
- *waitpid*( ) with a *pid* argument less than or equal to 0 or equal to the process ID of the command line interpreter
- Any other function not defined in this volume of IEEE Std 1003.1-2001 that could do one of the above

In any case, *pclose*( ) shall not return before the child process created by *popen*( ) has terminated.

If the command language interpreter cannot be executed, the child termination status returned by *pclose*( ) shall be as if the command language interpreter terminated using *exit*(127) or *_exit*(127).

The *pclose*( ) function shall not affect the termination status of any child of the calling process other than the one created by *popen*( ) for the associated stream.

If the argument *stream* to *pclose*( ) is not a pointer to a stream created by *popen*( ), the result of *pclose*( ) is undefined.

RETURN VALUE

Upon successful return, *pclose*( ) shall return the termination status of the command language interpreter. Otherwise, *pclose*( ) shall return –1 and set *errno* to indicate the error.

ERRORS

The *pclose*( ) function shall fail if:

[ECHILD] The status of the child process could not be obtained, as described above.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

There is a requirement that *pclose*( ) not return before the child process terminates. This is intended to disallow implementations that return [EINTR] if a signal is received while waiting. If *pclose*( ) returned before the child terminated, there would be no way for the application to discover which child used to be associated with the stream, and it could not do the cleanup itself.

If the stream pointed to by *stream* was not created by *popen*( ), historical implementations of *pclose*( ) return –1 without setting *errno*. To avoid requiring *pclose*( ) to set *errno* in this case, IEEE Std 1003.1-2001 makes the behavior unspecified. An application should not use *pclose*( ) to

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

close any stream that was not created by *popen*( ).

Some historical implementations of *pclose*( ) either block or ignore the signals SIGINT, SIGQUIT, and SIGHUP while waiting for the child process to terminate. Since this behavior is not described for the *pclose*( ) function in IEEE Std 1003.1-2001, such implementations are not conforming. Also, some historical implementations return [EINTR] if a signal is received, even though the child process has not terminated. Such implementations are also considered non-conforming.

Consider, for example, an application that uses:

```
popen("command", "r")
```

to start *command*, which is part of the same application. The parent writes a prompt to its standard output (presumably the terminal) and then reads from the *popen*( )ed stream. The child reads the response from the user, does some transformation on the response (pathname expansion, perhaps) and writes the result to its standard output. The parent process reads the result from the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that both processes do appropriate buffer flushing, this would be expected to work.

To conform to IEEE Std 1003.1-2001, *pclose*( ) must use *waitpid*( ), or some similar function, instead of *wait*( ).

The code sample below illustrates how the *pclose*( ) function might be implemented on a system conforming to IEEE Std 1003.1-2001.

```
int pclose(FILE *stream)
{
    int stat;
    pid_t pid;

    pid = <pid for process created for stream by popen()>
    (void) fclose(stream);
    while (waitpid(pid, &stat, 0) == -1) {
        if (errno != EINTR){
            stat = -1;
            break;
        }
    }
    return(stat);
}
```

FUTURE DIRECTIONS

None.

SEE ALSO

fork( ), *popen*( ), *waitpid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

perror — write error messages to standard error

SYNOPSIS

```
#include <stdio.h>

void perror(const char *s);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *perror*( ) function shall map the error number accessed through the symbol *errno* to a language-dependent error message, which shall be written to the standard error stream as follows:

- First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the string pointed to by *s* followed by a colon and a <space>.
- Then an error message string followed by a <newline>.

The contents of the error message strings shall be the same as those returned by *strerror*( ) with argument *errno*.

CX The *perror*( ) function shall mark the file associated with the standard error stream as having been written (*st_ctime*, *st_mtime* marked for update) at some time between its successful completion and *exit*( ), *abort*( ), or the completion of *fflush*( ) or *fclose*( ) on *stderr*.

The *perror*( ) function shall not change the orientation of the standard error stream.

RETURN VALUE

The *perror*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

Printing an Error Message for a Function

The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs, the *perror*( ) function prints a message and the program exits.

```
#include <stdio.h>
#include <stdlib.h>
...
char *bufptr;
size_t szbuf;
...
if ((bufptr = malloc(szbuf)) == NULL) {
    perror("malloc"); exit(2);
}
...
```

APPLICATION USAGE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

-:-`,,,`-`-`-,,-,-,`,,`-`-`--

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

27824 **RATIONALE**

27825 None.

27826 **FUTURE DIRECTIONS**

27827 None.

27828 **SEE ALSO**

27829 *strerror*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

27830 **CHANGE HISTORY**

27831 First released in Issue 1. Derived from Issue 1 of the SVID.

27832 **Issue 5**

27833 A paragraph is added to the DESCRIPTION indicating that *perror*( ) does not change the
27834 orientation of the standard error stream.

27835 **Issue 6**

27836 Extensions beyond the ISO C standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27837 **NAME**

27838 pipe — create an interprocess channel

27839 **SYNOPSIS**

27840 `#include <unistd.h>`

27841 `int pipe(int fildes[2]);`

27842 **DESCRIPTION**

27843 The *pipe*( ) function shall create a pipe and place two file descriptors, one each into the
27844 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write
27845 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe*( )
27846 call. The O_NONBLOCK and FD_CLOEXEC flags shall be clear on both file descriptors. (The
27847 *fcntl*( ) function can be used to set both these flags.)

27848 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A
27849 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a
27850 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether
27851 *fildes*[1] is also open for reading.

27852 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open
27853 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

27854 Upon successful completion, *pipe*( ) shall mark for update the *st_atime*, *st_ctime*, and *st_mtime*
27855 fields of the pipe.

27856 **RETURN VALUE**

27857 Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to
27858 indicate the error.

27859 **ERRORS**

27860 The *pipe*( ) function shall fail if:

27861 [EMFILE] More than {OPEN_MAX} minus two file descriptors are already in use by this
27862 process.

27863 [ENFILE] The number of simultaneously open files in the system would exceed a
27864 system-imposed limit.

27865 **EXAMPLES**

27866 None.

27867 **APPLICATION USAGE**

27868 None.

27869 **RATIONALE**

27870 The wording carefully avoids using the verb ‘‘to open’’ in order to avoid any implication of use
27871 of *open*( ); see also *write*( ).

27872 **FUTURE DIRECTIONS**

27873 None.

27874 **SEE ALSO**

27875 *fcntl*( ), *read*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**,
27876 **<unistd.h>**

27877 **CHANGE HISTORY**

27878 First released in Issue 1. Derived from Issue 1 of the SVID.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27879 **Issue 6**

27880 The following new requirements on POSIX implementations derive from alignment with the
27881 Single UNIX Specification:

27882 • The DESCRIPTION is updated to indicate that certain dispositions of *fildes*[0] and *fildes*[1]
27883 are unspecified.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

poll — input/output multiplexing

SYNOPSIS

XSI

```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

DESCRIPTION

The *poll*( ) function provides applications with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by *fds*, *poll*( ) shall examine the given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the *fds* array is specified by *nfds*. The *poll*( ) function shall identify those file descriptors on which an application can read or write data, or on which certain events have occurred.

The *fds* argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one member for each open file descriptor of interest. The array's members are **pollfd** structures within which *fd* specifies an open file descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the following event flags:

POLLIN	Data other than high-priority data may be read without blocking.
XSR	For STREAMS, this flag is set in *revents* even if the message is of zero length. This flag shall be equivalent to POLLRDNORM | POLLRDBAND.
POLLRDNORM	Normal data may be read without blocking.
XSR	For STREAMS, data on priority band 0 may be read without blocking. This flag is set in *revents* even if the message is of zero length.
POLLRDBAND	Priority data may be read without blocking.
XSR	For STREAMS, data on priority bands greater than 0 may be read without blocking. This flag is set in *revents* even if the message is of zero length.
POLLPRI	High-priority data may be read without blocking.
XSR	For STREAMS, this flag is set in *revents* even if the message is of zero length.
POLLOUT	Normal data may be written without blocking.
XSR	For STREAMS, data on priority band 0 may be written without blocking.
POLLWRNORM	Equivalent to POLLOUT.
POLLWRBAND	Priority data may be written.
XSR	For STREAMS, data on priority bands greater than 0 may be written without blocking. If any priority band has been written to on this STREAM, this event only examines bands that have been written to at least once.
POLLERR	An error has occurred on the device or stream. This flag is only valid in the *revents* bitmask; it shall be ignored in the *events* member.
POLLHUP	The device has been disconnected. This event and POLLOUT are mutually-exclusive; a stream can never be writable if a hangup has occurred. However, this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are not mutually-exclusive. This flag is only valid in the *revents* bitmask; it shall be ignored in the *events* member.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27926 POLLNVAL The specified *fd* value is invalid. This flag is only valid in the *revents* member;
27927 it shall ignored in the *events* member.

27928 The significance and semantics of normal, priority, and high-priority data are file and device-
27929 specific.

27930 If the value of *fd* is less than 0, *events* shall be ignored, and *revents* shall be set to 0 in that entry on
27931 return from *poll*( ).

27932 In each **pollfd** structure, *poll*( ) shall clear the *revents* member, except that where the application
27933 requested a report on a condition by setting one of the bits of *events* listed above, *poll*( ) shall set
27934 the corresponding bit in *revents* if the requested condition is true. In addition, *poll*( ) shall set the
27935 POLLHUP, POLLERR, and POLLNVAL flag in *revents* if the condition is true, even if the
27936 application did not set the corresponding bit in *events*.

27937 If none of the defined events have occurred on any selected file descriptor, *poll*( ) shall wait at
27938 least *timeout* milliseconds for an event to occur on any of the selected file descriptors. If the value
27939 of *timeout* is 0, *poll*( ) shall return immediately. If the value of *timeout* is –1, *poll*( ) shall block until
27940 a requested event occurs or until the call is interrupted.

27941 Implementations may place limitations on the granularity of timeout intervals. If the requested
27942 timeout interval requires a finer granularity than the implementation supports, the actual
27943 timeout interval shall be rounded up to the next supported value.

27944 The *poll*( ) function shall not be affected by the O_NONBLOCK flag.

27945 The *poll*( ) function shall support regular files, terminal and pseudo-terminal devices, FIFOs,
27946 XSR pipes, sockets and STREAMS-based files. The behavior of *poll*( ) on elements of *fds* that refer to
27947 other types of file is unspecified.

27948 Regular files shall always poll TRUE for reading and writing.

27949 A file descriptor for a socket that is listening for connections shall indicate that it is ready for
27950 reading, once connections are available. A file descriptor for a socket that is connecting
27951 asynchronously shall indicate that it is ready for writing, once a connection has been established.

27952 **RETURN VALUE**

27953 Upon successful completion, *poll*( ) shall return a non-negative value. A positive value indicates
27954 the total number of file descriptors that have been selected (that is, file descriptors for which the
27955 *revents* member is non-zero). A value of 0 indicates that the call timed out and no file descriptors
27956 have been selected. Upon failure, *poll*( ) shall return –1 and set *errno* to indicate the error.

27957 **ERRORS**

27958 The *poll*( ) function shall fail if:

27959 [EAGAIN] The allocation of internal data structures failed but a subsequent request may
27960 succeed.

27961 [EINTR] A signal was caught during *poll*( ).

27962 XSR [EINVAL] The *nfds* argument is greater than {OPEN_MAX}, or one of the *fd* members
27963 refers to a STREAM or multiplexer that is linked (directly or indirectly)
27964 downstream from a multiplexer.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Checking for Events on a Stream

The following example opens a pair of STREAMS devices and then waits for either one to become writable. This example proceeds as follows:

1. Sets the *timeout* parameter to 500 milliseconds.
2. Opens the STREAMS devices **/dev/dev0** and **/dev/dev1**, and then polls them, specifying POLLOUT and POLLWRBAND as the events of interest.

   The STREAMS device names **/dev/dev0** and **/dev/dev1** are only examples of how STREAMS devices can be named; STREAMS naming conventions may vary among systems conforming to the IEEE Std 1003.1-2001.
3. Uses the *ret* variable to determine whether an event has occurred on either of the two STREAMS. The *poll*( ) function is given 500 milliseconds to wait for an event to occur (if it has not occurred prior to the *poll*( ) call).
4. Checks the returned value of *ret*. If a positive value is returned, one of the following can be done:
   a. Priority data can be written to the open STREAM on priority bands greater than 0, because the POLLWRBAND event occurred on the open STREAM (*fds*[0] or *fds*[1]).
   b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT event occurred on the open STREAM (*fds*[0] or *fds*[1]).
5. If the returned value is not a positive value, permission to write data to the open STREAM (on any priority band) is denied.
6. If the POLLHUP event occurs on the open STREAM (*fds*[0] or *fds*[1]), the device on the open STREAM has disconnected.

```
#include <stropts.h>
#include <poll.h>
...
struct pollfd fds[2];
int timeout_msecs = 500;
int ret;
    int i;

/* Open STREAMS device. */
fds[0].fd = open("/dev/dev0", ...);
fds[1].fd = open("/dev/dev1", ...);
    fds[0].events = POLLOUT | POLLWRBAND;
    fds[1].events = POLLOUT | POLLWRBAND;

ret = poll(fds, 2, timeout_msecs);

if (ret > 0) {
    /* An event on one of the fds has occurred. */
    for (i=0; i<2; i++) {
        if (fds[i].revents & POLLWRBAND) {
        /* Priority data may be written on device number i. */
...
        }
        if (fds[i].revents & POLLOUT) {
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
                /* Data may be written on device number i. */
        ...
                }
                if (fds[i].revents & POLLHUP) {
                /* A hangup has occurred on device number i. */
        ...
                }
            }
        }
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 38), *getmsg*( ), *putmsg*( ), *read*( ), *select*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<poll.h>**, **<stropts.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The description of POLLWRBAND is updated.

Issue 6

Text referring to sockets is added to the DESCRIPTION.

Text relating to the XSI STREAMS Option Group is marked.

The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of POLLWRBAND.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

popen — initiate pipe streams to or from a process

SYNOPSIS

CX
```
#include <stdio.h>

FILE *popen(const char *command, const char *mode);
```

DESCRIPTION

The *popen*( ) function shall execute the command specified by the string *command*. It shall create a pipe between the calling program and the executed command, and shall return a pointer to a stream that can be used to either read from or write to the pipe.

The environment of the executed command shall be as if a child process were created within the *popen*( ) call using the *fork*( ) function, and the child invoked the *sh* utility using the call:

```
execl(shell path, "sh", "-c", command, (char *)0);
```

where *shell path* is an unspecified pathname for the *sh* utility.

The *popen*( ) function shall ensure that any streams from previous *popen*( ) calls that remain open in the parent process are closed in the new child process.

The *mode* argument to *popen*( ) is a string that specifies I/O mode:

1. If *mode* is *r*, when the child process is started, its file descriptor STDOUT_FILENO shall be the writable end of the pipe, and the file descriptor *fileno*(*stream*) in the calling process, where *stream* is the stream pointer returned by *popen*( ), shall be the readable end of the pipe.
2. If *mode* is *w*, when the child process is started its file descriptor STDIN_FILENO shall be the readable end of the pipe, and the file descriptor *fileno*(*stream*) in the calling process, where *stream* is the stream pointer returned by *popen*( ), shall be the writable end of the pipe.
3. If *mode* is any other value, the result is undefined.

After *popen*( ), both the parent and the child process shall be capable of executing independently before either terminates.

Pipe streams are byte-oriented.

RETURN VALUE

Upon successful completion, *popen*( ) shall return a pointer to an open stream that can be used to read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate the error.

ERRORS

The *popen*( ) function may fail if:

[EMFILE] {FOPEN_MAX} or {STREAM_MAX} streams are currently open in the calling process.

[EINVAL] The *mode* argument is invalid.

The *popen*( ) function may also set *errno* values as described by *fork*( ) or *pipe*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

Since open files are shared, a mode *r* command can be used as an input filter and a mode *w* command as an output filter.

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be prevented by careful buffer flushing; for example, with *fflush*( ).

A stream opened by *popen*( ) should be closed by *pclose*( ).

The behavior of *popen*( ) is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and *wb* might be supported by specific implementations, but these would not be portable features. Note that historical implementations of *popen*( ) only check to see if the first character of *mode* is *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be treated as *mode w*.

If the application calls *waitpid*( ) or *waitid*( ) with a *pid* argument greater than 0, and it still has a stream that was called with *popen*( ) open, it must ensure that *pid* does not refer to the process started by *popen*( ).

To determine whether or not the environment specified in the Shell and Utilities volume of IEEE Std 1003.1-2001 is present, use the function call:

```
sysconf(_SC_2_VERSION)
```

(See *sysconf*( )).

RATIONALE

The *popen*( ) function should not be used by programs that have set user (or group) ID privileges. The *fork*( ) and *exec* family of functions (except *execlp*( ) and *execvp*( )), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

If the original and *popen*( )ed processes both intend to read or write or read and write a common file, and either will be using FILE-type C functions (*fread*( ), *fwrite*( ), and so on), the rules for sharing file handles must be observed (see Section 2.5.1 (on page 35)).

FUTURE DIRECTIONS

None.

SEE ALSO

pclose( ), *pipe*( ), *sysconf*( ), *system*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, *sh*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional [EMFILE] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_fadvise — file advisory information (**ADVANCED REALTIME**)

SYNOPSIS

ADV
```
#include <fcntl.h>

int posix_fadvise(int fd, off_t offset, size_t len, int advice);
```

DESCRIPTION

The *posix_fadvise*( ) function shall advise the implementation on the expected behavior of the application with respect to the data in the file associated with the open file descriptor, *fd*, starting at *offset* and continuing for *len* bytes. The specified range need not currently exist in the file. If *len* is zero, all data following *offset* is specified. The implementation may use this information to optimize handling of the specified data. The *posix_fadvise*( ) function shall have no effect on the semantics of other operations on the specified data, although it may affect the performance of other operations.

The advice to be applied to the data is specified by the *advice* parameter and may be one of the following values:

POSIX_FADV_NORMAL
: Specifies that the application has no advice to give on its behavior with respect to the specified data. It is the default characteristic if no advice is given for an open file.

POSIX_FADV_SEQUENTIAL
: Specifies that the application expects to access the specified data sequentially from lower offsets to higher offsets.

POSIX_FADV_RANDOM
: Specifies that the application expects to access the specified data in a random order.

POSIX_FADV_WILLNEED
: Specifies that the application expects to access the specified data in the near future.

POSIX_FADV_DONTNEED
: Specifies that the application expects that it will not access the specified data in the near future.

POSIX_FADV_NOREUSE
: Specifies that the application expects to access the specified data once and then not reuse it thereafter.

These values are defined in **<fcntl.h>**.

RETURN VALUE

Upon successful completion, *posix_fadvise*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_fadvise*( ) function shall fail if:

[EBADF] The *fd* argument is not a valid file descriptor.

[EINVAL] The value of *advice* is invalid.

[ESPIPE] The *fd* argument is associated with a pipe or FIFO.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

28159 **EXAMPLES**

28160 None.

28161 **APPLICATION USAGE**

28162 The *posix_fadvise*( ) function is part of the Advisory Information option and need not be provided
28163 on all implementations.

28164 **RATIONALE**

28165 None.

28166 **FUTURE DIRECTIONS**

28167 None.

28168 **SEE ALSO**

28169 *posix_madvise*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**

28170 **CHANGE HISTORY**

28171 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28172 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_fallocate — file space control (**ADVANCED REALTIME**)

SYNOPSIS

ADV
```
#include <fcntl.h>

int posix_fallocate(int fd, off_t offset, size_t len);
```

DESCRIPTION

The *posix_fallocate*( ) function shall ensure that any required storage for regular file data starting at *offset* and continuing for *len* bytes is allocated on the file system storage media. If *posix_fallocate*( ) returns successfully, subsequent writes to the specified file data shall not fail due to the lack of free space on the file system storage media.

If the *offset*+*len* is beyond the current file size, then *posix_fallocate*( ) shall adjust the file size to *offset*+*len*. Otherwise, the file size shall not be changed.

It is implementation-defined whether a previous *posix_fadvise*( ) call influences allocation strategy.

Space allocated via *posix_fallocate*( ) shall be freed by a successful call to *creat*( ) or *open*( ) that truncates the size of the file. Space allocated via *posix_fallocate*( ) may be freed by a successful call to *ftruncate*( ) that reduces the file size to a size smaller than *offset*+*len*.

RETURN VALUE

Upon successful completion, *posix_fallocate*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_fallocate*( ) function shall fail if:

[EBADF]	The *fd* argument is not a valid file descriptor.
[EBADF]	The *fd* argument references a file that was opened without write permission.
[EFBIG]	The value of *offset*+*len* is greater than the maximum file size.
[EINTR]	A signal was caught during execution.
[EINVAL]	The *len* argument was zero or the *offset* argument was less than zero.
[EIO]	An I/O error occurred while reading from or writing to a file system.
[ENODEV]	The *fd* argument does not refer to a regular file.
[ENOSPC]	There is insufficient free space remaining on the file system storage media.
[ESPIPE]	The *fd* argument is associated with a pipe or FIFO.

EXAMPLES

None.

APPLICATION USAGE

The *posix_fallocate*( ) function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

28212 **FUTURE DIRECTIONS**

28213 None.

28214 **SEE ALSO**

28215 *creat*( ), *ftruncate*( ), *open*( ), *unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
28216 **<fcntl.h>**

28217 **CHANGE HISTORY**

28218 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28219 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_madvise — memory advisory information and alignment control (**ADVANCED REALTIME**)

SYNOPSIS

ADV
```
#include <sys/mman.h>

int posix_madvise(void *addr, size_t len, int advice);
```

DESCRIPTION

MF|SHM The *posix_madvise*( ) function need only be supported if either the Memory Mapped Files or the Shared Memory Objects options are supported.

The *posix_madvise*( ) function shall advise the implementation on the expected behavior of the application with respect to the data in the memory starting at address *addr*, and continuing for *len* bytes. The implementation may use this information to optimize handling of the specified data. The *posix_madvise*( ) function shall have no effect on the semantics of access to memory in the specified range, although it may affect the performance of access.

The implementation may require that *addr* be a multiple of the page size, which is the value returned by *sysconf*( ) when the name value _SC_PAGESIZE is used.

The advice to be applied to the memory range is specified by the *advice* parameter and may be one of the following values:

POSIX_MADV_NORMAL
: Specifies that the application has no advice to give on its behavior with respect to the specified range. It is the default characteristic if no advice is given for a range of memory.

POSIX_MADV_SEQUENTIAL
: Specifies that the application expects to access the specified range sequentially from lower addresses to higher addresses.

POSIX_MADV_RANDOM
: Specifies that the application expects to access the specified range in a random order.

POSIX_MADV_WILLNEED
: Specifies that the application expects to access the specified range in the near future.

POSIX_MADV_DONTNEED
: Specifies that the application expects that it will not access the specified range in the near future.

These values are defined in the **<sys/mman.h>** header.

RETURN VALUE

Upon successful completion, *posix_madvise*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_madvise*( ) function shall fail if:

[EINVAL] The value of *advice* is invalid.

[ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly or completely outside the range allowed for the address space of the calling process.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *posix_madvise*( ) function may fail if:

[EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf*( ) when the name value _SC_PAGESIZE is used.

[EINVAL] The value of *len* is zero.

EXAMPLES

None.

APPLICATION USAGE

The *posix_madvise*( ) function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *posix_fadvise*( ), *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #102 is applied.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_mem_offset — find offset and length of a mapped typed memory block (**ADVANCED REALTIME**)

SYNOPSIS

TYM
```
#include <sys/mman.h>

int posix_mem_offset(const void *restrict addr, size_t len,
    off_t *restrict off, size_t *restrict contig_len,
    int *restrict fildes);
```

DESCRIPTION

The *posix_mem_offset*( ) function shall return in the variable pointed to by *off* a value that identifies the offset (or location), within a memory object, of the memory block currently mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used (via *mmap*( )) to establish the mapping which contains *addr*. If that descriptor was closed since the mapping was established, the returned value of *fildes* shall be –1. The *len* argument specifies the length of the block of the memory object the user wishes the offset for; upon return, the value pointed to by *contig_len* shall equal either *len*, or the length of the largest contiguous block of the memory object that is currently mapped to the calling process starting at *addr*, whichever is smaller.

If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig_len* values obtained by calling *posix_mem_offset*( ) are used in a call to *mmap*( ) with a file descriptor that refers to the same memory pool as *fildes* (either through the same port or through a different port), and that was opened with neither the POSIX_TYPED_MEM_ALLOCATE nor the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, the typed memory area that is mapped shall be exactly the same area that was mapped at *addr* in the address space of the process that called *posix_mem_offset*( ).

If the memory object specified by *fildes* is not a typed memory object, then the behavior of this function is implementation-defined.

RETURN VALUE

Upon successful completion, the *posix_mem_offset*( ) function shall return zero; otherwise, the corresponding error status value shall be returned.

ERRORS

The *posix_mem_offset*( ) function shall fail if:

[EACCES] The process has not mapped a memory object supported by this function at the given address *addr*.

This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

28323 **FUTURE DIRECTIONS**

28324 None.

28325 **SEE ALSO**

28326 *mmap*( ), *posix_typed_mem_open*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
28327 **<sys/mman.h>**

28328 **CHANGE HISTORY**

28329 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

28330 **NAME**

28331 posix_memalign — aligned memory allocation (**ADVANCED REALTIME**)

28332 **SYNOPSIS**

28333 ADV
```
#include <stdlib.h>
```

28334
```
int posix_memalign(void **memptr, size_t alignment, size_t size);
```

28335

28336 **DESCRIPTION**

28337 The *posix_memalign*( ) function shall allocate *size* bytes aligned on a boundary specified by
28338 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*
28339 shall be a multiple of *sizeof*(**void ***), that is also a power of two. Upon successful completion, the
28340 value pointed to by *memptr* shall be a multiple of *alignment*.

28341 CX The *free*( ) function shall deallocate memory that has previously been allocated by
28342 *posix_memalign*( ).

28343 **RETURN VALUE**

28344 Upon successful completion, *posix_memalign*( ) shall return zero; otherwise, an error number
28345 shall be returned to indicate the error.

28346 **ERRORS**

28347 The *posix_memalign*( ) function shall fail if:

28348 [EINVAL] The value of the alignment parameter is not a power of two multiple of
28349 *sizeof*(**void ***).

28350 [ENOMEM] There is insufficient memory available with the requested alignment.

28351 **EXAMPLES**

28352 None.

28353 **APPLICATION USAGE**

28354 The *posix_memalign*( ) function is part of the Advisory Information option and need not be
28355 provided on all implementations.

28356 **RATIONALE**

28357 None.

28358 **FUTURE DIRECTIONS**

28359 None.

28360 **SEE ALSO**

28361 *free*( ), *malloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

28362 **CHANGE HISTORY**

28363 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28364 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_openpt — open a pseudo-terminal device

SYNOPSIS

XSI
```
#include <stdlib.h>
#include <fcntl.h>

int posix_openpt(int oflag);
```

DESCRIPTION

The *posix_openpt*( ) function shall establish a connection between a master device for a pseudo-terminal and a file descriptor. The file descriptor is used by other I/O functions that refer to that pseudo-terminal.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in **<fcntl.h>**:

O_RDWR Open for reading and writing.

O_NOCTTY If set *posix_openpt*( ) shall not cause the terminal device to become the controlling terminal for the process.

The behavior of other values for the *oflag* argument is unspecified.

RETURN VALUE

Upon successful completion, the *posix_openpt*( ) function shall open a master pseudo-terminal device and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *posix_openpt*( ) function shall fail if:

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENFILE] The maximum allowable number of files is currently open in the system.

The *posix_openpt*( ) function may fail if:

[EINVAL] The value of *oflag* is not valid.

[EAGAIN] Out of pseudo-terminal resources.

XSR [ENOSR] Out of STREAMS resources.

EXAMPLES

Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File Descriptor

```
#include <fcntl.h>
#include <stdio.h>

int masterfd, slavefd;
char *slavedevice;

masterfd = posix_openpt(O_RDWR|O_NOCTTY);

if (masterfd == -1
    || grantpt (masterfd) == -1
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
28406           || unlockpt (masterfd) == -1
28407           || (slavedevice = ptsname (masterfd)) == NULL)
28408           return -1;

28409       printf("slave device is: %s\n", slavedevice);

28410       slavefd = open(slave, O_RDWR|O_NOCTTY);
28411       if (slavefd < 0)
28412           return -1;
```

28413 **APPLICATION USAGE**

28414 This function is a method for portably obtaining a file descriptor of a master terminal device for
28415 a pseudo-terminal. The *grantpt*( ) and *ptsname*( ) functions can be used to manipulate mode and
28416 ownership permissions, and to obtain the name of the slave device, respectively.

28417 **RATIONALE**

28418 The standard developers considered the matter of adding a special device for cloning master
28419 pseudo-terminals: the **/dev/ptmx** device. However, consensus could not be reached, and it was
28420 felt that adding a new function would permit other implementations. The *posix_openpt*( )
28421 function is designed to complement the *grantpt*( ), *ptsname*( ), and *unlockpt*( ) functions.

28422 On implementations supporting the **/dev/ptmx** clone device, opening the master device of a
28423 pseudo-terminal is simply:

```
28424       mfdp = open("/dev/ptmx", oflag );
28425       if (mfdp < 0)
28426           return -1;
```

28427 **FUTURE DIRECTIONS**

28428 None.

28429 **SEE ALSO**

28430 *grantpt*( ), *open*( ), *ptsname*( ), *unlockpt*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
28431 **<fcntl.h>**

28432 **CHANGE HISTORY**

28433 First released in Issue 6.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_spawn, posix_spawnp — spawn a process (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn(pid_t *restrict pid, const char *restrict path,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
int posix_spawnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char * const envp[restrict]);
```

DESCRIPTION

The *posix_spawn*( ) and *posix_spawnp*( ) functions shall create a new process (child process) from the specified process image. The new process image shall be constructed from a regular executable file called the new process image file.

When a C program is executed as the result of this call, it shall be entered as a C-language function call as follows:

```
int main(int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
extern char **environ;
```

shall be initialized as a pointer to an array of character pointers to the environment strings.

The argument *argv* is an array of character pointers to null-terminated strings. The last member of this array shall be a null pointer and is not counted in *argc*. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process image being started by the *posix_spawn*( ) or *posix_spawnp*( ) function.

The argument *envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The environment array is terminated by a null pointer.

The number of bytes available for the child process' combined argument and environment lists is {ARG_MAX}. The implementation shall specify in the system documentation (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance) whether any list overhead, such as length words, null terminators, pointers, or alignment bytes, is included in this total.

The *path* argument to *posix_spawn*( ) is a pathname that identifies the new process image file to execute.

The *file* parameter to *posix_spawnp*( ) shall be used to construct a pathname that identifies the new process image file. If the *file* parameter contains a slash character, the *file* parameter shall be used as the pathname for the new process image file. Otherwise, the path prefix for this file shall be obtained by a search of the directories passed as the environment variable *PATH* (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If this environment variable is not defined, the results of the search are implementation-defined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If *file_actions* is a null pointer, then file descriptors open in the calling process shall remain open in the child process, except for those whose close-on-*exec* flag FD_CLOEXEC is set (see *fcntl*( )). For those file descriptors that remain open, all attributes of the corresponding open file descriptions, including file locks (see *fcntl*( )), shall remain unchanged.

If *file_actions* is not NULL, then the file descriptors open in the child process shall be those open in the calling process as modified by the spawn file actions object pointed to by *file_actions* and the FD_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have been processed. The effective order of processing the spawn file actions shall be:

1. The set of open file descriptors for the child process shall initially be the same set as is open for the calling process. All attributes of the corresponding open file descriptions, including file locks (see *fcntl*( )), shall remain unchanged.
2. The signal mask, signal default actions, and the effective user and group IDs for the child process shall be changed as specified in the attributes object referenced by *attrp*.
3. The file actions specified by the spawn file actions object shall be performed in the order in which they were added to the spawn file actions object.
4. Any file descriptor that has its FD_CLOEXEC flag set (see *fcntl*( )) shall be closed.

The **posix_spawnattr_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at least the attributes defined below.

If the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced by *attrp*.

As a special case, if the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero, then the child shall be in a new process group with a process group ID equal to its process ID.

If the POSIX_SPAWN_SETPGROUP flag is not set in the *spawn-flags* attribute of the object referenced by *attrp*, the new child process shall inherit the parent's process group.

PS If the POSIX_SPAWN_SETSCHEDPARAM flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, but POSIX_SPAWN_SETSCHEDULER is not set, the new process image shall initially have the scheduling policy of the calling process with the scheduling parameters specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSCHEDULER flag is set in the *spawn-flags* attribute of the object referenced by *attrp* (regardless of the setting of the POSIX_SPAWN_SETSCHEDPARAM flag), the new process image shall initially have the scheduling policy specified in the *spawn-schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in the *spawn-schedparam* attribute of the same object.

The POSIX_SPAWN_RESETIDS flag in the *spawn-flags* attribute of the object referenced by *attrp* governs the effective user ID of the child process. If this flag is not set, the child process shall inherit the parent process' effective user ID. If this flag is set, the child process' effective user ID shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the child process shall become that file's owner ID before the new process image begins execution.

The POSIX_SPAWN_RESETIDS flag in the *spawn-flags* attribute of the object referenced by *attrp* also governs the effective group ID of the child process. If this flag is not set, the child process shall inherit the parent process' effective group ID. If this flag is set, the child process' effective group ID shall be reset to the parent's real group ID. In either case, if the set-group-ID mode bit

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

of the new process image file is set, the effective group ID of the child process shall become that file's group ID before the new process image begins execution.

If the POSIX_SPAWN_SETSIGMASK flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-sigmask* attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSIGDEF flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to their default actions in the child process. Signals set to the default action in the parent process shall be set to the default action in the child process.

Signals set to be caught by the calling process shall be set to the default action in the child process.

Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn-flags* attribute of the object referenced by *attrp* and the signals being indicated in the *spawn-sigdefault* attribute of the object referenced by *attrp*.

If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn_flags* attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the *spawn_sigdefault* attribute of the object referenced by *attrp*.

If the value of the *attrp* pointer is NULL, then the default values are used.

All process attributes, other than those influenced by the attributes set in the object referenced by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, shall appear in the new process image as though *fork*( ) had been called to create a child process and then a member of the *exec* family of functions had been called by the child process to execute the new process image.

THR It is implementation-defined whether the fork handlers are run when *posix_spawn*( ) or *posix_spawnp*( ) is called.

RETURN VALUE

Upon successful completion, *posix_spawn*( ) and *posix_spawnp*( ) shall return the process ID of the child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and shall return zero as the function return value. Otherwise, no child process shall be created, the value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number shall be returned as the function return value to indicate the error. If the *pid* argument is a null pointer, the process ID of the child is not returned to the caller.

ERRORS

The *posix_spawn*( ) and *posix_spawnp*( ) functions may fail if:

[EINVAL] The value specified by *file_actions* or *attrp* is invalid.

If this error occurs after the calling process successfully returns from the *posix_spawn*( ) or *posix_spawnp*( ) function, the child process may exit with exit status 127.

If *posix_spawn*( ) or *posix_spawnp*( ) fail for any of the reasons that would cause *fork*( ) or one of the *exec* family of functions to fail, an error value shall be returned as described by *fork*( ) and *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute of the object referenced by *attrp*, and *posix_spawn*( ) or *posix_spawnp*( ) fails while changing the child's process group, an error value shall be returned as described by *setpgid*( ) (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

PS If POSIX_SPAWN_SETSCHEDPARAM is set and POSIX_SPAWN_SETSCHEDULER is not set in the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix_spawn*( ) or *posix_spawnp*( ) fails for any of the reasons that would cause *sched_setparam*( ) to fail, an error value shall be returned as described by *sched_setparam*( ) (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If POSIX_SPAWN_SETSCHEDULER is set in the *spawn-flags* attribute of the object referenced by *attrp*, and if *posix_spawn*( ) or *posix_spawnp*( ) fails for any of the reasons that would cause *sched_setscheduler*( ) to fail, an error value shall be returned as described by *sched_setscheduler*( ) (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If the *file_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be performed, and if *posix_spawn*( ) or *posix_spawnp*( ) fails for any of the reasons that would cause *close*( ), *dup2*( ), or *open*( ) to fail, an error value shall be returned as described by *close*( ), *dup2*( ), and *open*( ), respectively (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127). An open file action may, by itself, result in any of the errors described by *close*( ) or *dup2*( ), in addition to those described by *open*( ).

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

The *posix_spawn*( ) function and its close relation *posix_spawnp*( ) have been introduced to overcome the following perceived difficulties with *fork*( ): the *fork*( ) function is difficult or impossible to implement without swapping or dynamic address translation.

- Swapping is generally too slow for a realtime environment.
- Dynamic address translation is not available everywhere that POSIX might be useful.
- Processes are too useful to simply option out of POSIX whenever it must run without address translation or other MMU services.

Thus, POSIX needs process creation and file execution primitives that can be efficiently implemented without address translation or other MMU services.

The *posix_spawn*( ) function is implementable as a library routine, but both *posix_spawn*( ) and *posix_spawnp*( ) are designed as kernel operations. Also, although they may be an efficient replacement for many *fork*( )/*exec* pairs, their goal is to provide useful process creation primitives for systems that have difficulty with *fork*( ), not to provide drop-in replacements for *fork*( )/*exec*.

This view of the role of *posix_spawn*( ) and *posix_spawnp*( ) influenced the design of their API. It does not attempt to provide the full functionality of *fork*( )/*exec* in which arbitrary user-specified operations of any sort are permitted between the creation of the child process and the execution of the new process image; any attempt to reach that level would need to provide a programming language as parameters. Instead, *posix_spawn*( ) and *posix_spawnp*( ) are process creation primitives like the *Start_Process* and *Start_Process_Search* Ada language bindings package *POSIX_Process_Primitives* and also like those in many operating systems that are not UNIX

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

systems, but with some POSIX-specific additions.

To achieve its coverage goals, *posix_spawn*( ) and *posix_spawnp*( ) have control of six types of inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and whether each signal ignored in the parent will remain ignored in the child, or be reset to its default action in the child.

Control of file descriptors is required to allow an independently written child process image to access data streams opened by and even generated or read by the parent process without being specifically coded to know which parent files and file descriptors are to be used. Control of the process group ID is required to control how the child process' job control relates to that of the parent.

Control of the signal mask and signal defaulting is sufficient to support the implementation of *system*( ). Although support for *system*( ) is not explicitly one of the goals for *posix_spawn*( ) and *posix_spawnp*( ), it is covered under the ''at least 50%'' coverage goal.

The intention is that the normal file descriptor inheritance across *fork*( ), the subsequent effect of the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec* family of functions should fully specify open file inheritance. The implementation need make no decisions regarding the set of open file descriptors when the child process image begins execution, those decisions having already been made by the caller and expressed as the set of open file descriptors and their FD_CLOEXEC flags at the time of the call and the spawn file actions object specified in the call. We have been assured that in cases where the POSIX *Start_Process* Ada primitives have been implemented in a library, this method of controlling file descriptor inheritance may be implemented very easily.

We can identify several problems with *posix_spawn*( ) and *posix_spawnp*( ), but there does not appear to be a solution that introduces fewer problems. Environment modification for child process attributes not specifiable via the *attrp* or *file_actions* arguments must be done in the parent process, and since the parent generally wants to save its context, it is more costly than similar functionality with *fork*( )/*exec*. It is also complicated to modify the environment of a multi-threaded process temporarily, since all threads must agree when it is safe for the environment to be changed. However, this cost is only borne by those invocations of *posix_spawn*( ) and *posix_spawnp*( ) that use the additional functionality. Since extensive modifications are not the usual case, and are particularly unlikely in time-critical code, keeping much of the environment control out of *posix_spawn*( ) and *posix_spawnp*( ) is appropriate design.

The *posix_spawn*( ) and *posix_spawnp*( ) functions do not have all the power of *fork*( )/*exec*. This is to be expected. The *fork*( ) function is a wonderfully powerful operation. We do not expect to duplicate its functionality in a simple, fast function with no special hardware requirements. It is worth noting that *posix_spawn*( ) and *posix_spawnp*( ) are very similar to the process creation operations on many operating systems that are not UNIX systems.

Requirements

The requirements for *posix_spawn*( ) and *posix_spawnp*( ) are:

- They must be implementable without an MMU or unusual hardware.
- They must be compatible with existing POSIX standards.

Additional goals are:

- They should be efficiently implementable.
- They should be able to replace at least 50% of typical executions of *fork*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- A system with *posix_spawn*( ) and *posix_spawnp*( ) and without *fork*( ) should be useful, at least for realtime applications.
- A system with *fork*( ) and the *exec* family should be able to implement *posix_spawn*( ) and *posix_spawnp*( ) as library routines.

Two-Syntax

POSIX *exec* has several calling sequences with approximately the same functionality. These appear to be required for compatibility with existing practice. Since the existing practice for the *posix_spawn**( ) functions is otherwise substantially unlike POSIX, we feel that simplicity outweighs compatibility. There are, therefore, only two names for the *posix_spawn**( ) functions.

The parameter list does not differ between *posix_spawn*( ) and *posix_spawnp*( ); *posix_spawnp*( ) interprets the second parameter more elaborately than *posix_spawn*( ).

Compatibility with POSIX.5 (Ada)

The *Start_Process* and *Start_Process_Search* procedures from the *POSIX_Process_Primitives* package from the Ada language binding to POSIX.1 encapsulate *fork*( ) and *exec* functionality in a manner similar to that of *posix_spawn*( ) and *posix_spawnp*( ). Originally, in keeping with our simplicity goal, the standard developers had limited the capabilities of *posix_spawn*( ) and *posix_spawnp*( ) to a subset of the capabilities of *Start_Process* and *Start_Process_Search*; certain non-default capabilities were not supported. However, based on suggestions by the ballot group to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings working group member, the standard developers decided that *posix_spawn*( ) and *posix_spawnp*( ) should be sufficiently powerful to implement *Start_Process* and *Start_Process_Search*. The rationale is that if the Ada language binding to such a primitive had already been approved as an IEEE standard, there can be little justification for not approving the functionally-equivalent parts of a C binding. The only three capabilities provided by *posix_spawn*( ) and *posix_spawnp*( ) that are not provided by *Start_Process* and *Start_Process_Search* are optionally specifying the child's process group ID, the set of signals to be reset to default signal handling in the child process, and the child's scheduling policy and parameters.

For the Ada language binding for *Start_Process* to be implemented with *posix_spawn*( ), that binding would need to explicitly pass an empty signal mask and the parent's environment to *posix_spawn*( ) whenever the caller of *Start_Process* allowed these arguments to default, since *posix_spawn*( ) does not provide such defaults. The ability of *Start_Process* to mask user-specified signals during its execution is functionally unique to the Ada language binding and must be dealt with in the binding separately from the call to *posix_spawn*( ).

Process Group

The process group inheritance field can be used to join the child process with an existing process group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*, the *setpgid*( ) mechanism will place the child process in a new process group.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

28696 **Threads**

28697 Without the *posix_spawn*( ) and *posix_spawnp*( ) functions, systems without address translation
28698 can still use threads to give an abstraction of concurrency. In many cases, thread creation
28699 suffices, but it is not always a good substitute. The *posix_spawn*( ) and *posix_spawnp*( ) functions
28700 are considerably ''heavier'' than thread creation. Processes have several important attributes that
28701 threads do not. Even without address translation, a process may have base-and-bound memory
28702 protection. Each process has a process environment including security attributes and file
28703 capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-
28704 uniform-memory-architecture multi-processors better than threads, and they are more
28705 convenient to use for activities that are not closely linked.

28706 The *posix_spawn*( ) and *posix_spawnp*( ) functions may not bring support for multiple processes to
28707 every configuration. Process creation is not the only piece of operating system support required
28708 to support multiple processes. The total cost of support for multiple processes may be quite high
28709 in some circumstances. Existing practice shows that support for multiple processes is
28710 uncommon and threads are common among ''tiny kernels''. There should, therefore, probably
28711 continue to be AEPs for operating systems with only one process.

28712 **Asynchronous Error Notification**

28713 A library implementation of *posix_spawn*( ) or *posix_spawnp*( ) may not be able to detect all
28714 possible errors before it forks the child process. IEEE Std 1003.1-2001 provides for an error
28715 indication returned from a child process which could not successfully complete the spawn
28716 operation via a special exit status which may be detected using the status value returned by
28717 *wait*( ) and *waitpid*( ).

28718 The *stat_val* interface and the macros used to interpret it are not well suited to the purpose of
28719 returning API errors, but they are the only path available to a library implementation. Thus, an
28720 implementation may cause the child process to exit with exit status 127 for any error detected
28721 during the spawn process after the *posix_spawn*( ) or *posix_spawnp*( ) function has successfully
28722 returned.

28723 The standard developers had proposed using two additional macros to interpret *stat_val*. The
28724 first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because
28725 of an error detected during the *posix_spawn*( ) or *posix_spawnp*( ) operations rather than during
28726 actual execution of the child process image; the second, WSPAWNERRNO, would have
28727 extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group
28728 strongly opposed this because it would make a library implementation of *posix_spawn*( ) or
28729 *posix_spawnp*( ) dependent on kernel modifications to *waitpid*( ) to be able to embed special
28730 information in *stat_val* to indicate a spawn failure.

28731 The 8 bits of child process exit status that are guaranteed by IEEE Std 1003.1-2001 to be
28732 accessible to the waiting parent process are insufficient to disambiguate a spawn error from any
28733 other kind of error that may be returned by an arbitrary process image. No other bits of the exit
28734 status are required to be visible in *stat_val*, so these macros could not be strictly implemented at
28735 the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use
28736 of this value by *system*( ) and *popen*( ) to signal failures in these operations that occur after the
28737 function has returned but before a shell is able to execute. The exit status of 127 does not
28738 uniquely identify this class of error, nor does it provide any detailed information on the nature
28739 of the failure. Note that a kernel implementation of *posix_spawn*( ) or *posix_spawnp*( ) is permitted
28740 (and encouraged) to return any possible error as the function value, thus providing more
28741 detailed failure information to the parent process.

28742 Thus, no special macros are available to isolate asynchronous *posix_spawn*( ) or *posix_spawnp*( )
28743 errors. Instead, errors detected by the *posix_spawn*( ) or *posix_spawnp*( ) operations in the context

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

of the child process before the new process image executes are reported by setting the child's exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the *stat_val* stored by the *wait*( ) or *waitpid*( ) functions to detect spawn failures to the extent that other status values with which the child process image may exit (before the parent can conclusively determine that the child process image has begun execution) are distinct from exit status 127.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *chmod*( ), *close*( ), *dup*( ), *exec*, *exit*( ), *fcntl*( ), *fork*( ), *kill*( ), *open*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_adddup2*( ), *posix_spawn_file_actions_addopen*( ), *posix_spawn_file_actions_destroy*( ), <REFERENCE UNDEFINED>(posix_spawn_file_actions_init), *posix_spawnattr_destroy*( ), *posix_spawnattr_init*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setflags*( ), *posix_spawnattr_setpgroup*( ), *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setschedpolicy*( ), *posix_spawnattr_setsigmask*( ), *sched_setparam*( ), *sched_setscheduler*( ), *setpgid*( ), *setuid*( ), *stat*( ), *times*( ), *wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are changed as well as the signal mask in step 2.

IEEE PASC Interpretation 1003.1 #132 is applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen — add close or open action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *
    file_actions, int fildes);
int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *
    restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```

DESCRIPTION

These functions shall add or delete a close or open action to a spawn file actions object.

A spawn file actions object is of type **posix_spawn_file_actions_t** (defined in **<spawn.h>**) and is used to specify a series of actions to be performed by a *posix_spawn*( ) or *posix_spawnp*( ) operation in order to arrive at the set of open file descriptors for the child process given the set of open file descriptors of the parent. IEEE Std 1003.1-2001 does not define comparison or assignment operators for the type **posix_spawn_file_actions_t**.

A spawn file actions object, when passed to *posix_spawn*( ) or *posix_spawnp*( ), shall specify how the set of open file descriptors in the calling process is transformed into a set of potentially open file descriptors for the spawned process. This transformation shall be as if the specified sequence of actions was performed exactly once, in the context of the spawned process (prior to execution of the new process image), in the order in which the actions were added to the object; additionally, when the new process image is executed, any file descriptor (from this new set) which has its FD_CLOEXEC flag set shall be closed (see *posix_spawn*( )).

The *posix_spawn_file_actions_addclose*( ) function shall add a *close* action to the object referenced by *file_actions* that shall cause the file descriptor *fildes* to be closed (as if *close*(*fildes*) had been called) when a new process is spawned using this file actions object.

The *posix_spawn_file_actions_addopen*( ) function shall add an *open* action to the object referenced by *file_actions* that shall cause the file named by *path* to be opened (as if *open*(*path*, *oflag*, *mode*) had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a new process is spawned using this file actions object. If *fildes* was already an open file descriptor, it shall be closed before the new file is opened.

The string described by *path* shall be copied by the *posix_spawn_file_actions_addopen*( ) function.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

[EBADF] The value specified by *fildes* is negative or greater than or equal to {OPEN_MAX}.

These functions may fail if:

[EINVAL] The value specified by *file_actions* is invalid.

[ENOMEM] Insufficient memory exists to add to the spawn file actions object.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

It shall not be considered an error for the *fildes* argument passed to these functions to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such error will be detected when the associated file actions object is later used during a *posix_spawn*( ) or *posix_spawnp*( ) operation.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

A spawn file actions object may be initialized to contain an ordered sequence of *close*( ), *dup2*( ), and *open*( ) operations to be used by *posix_spawn*( ) or *posix_spawnp*( ) to arrive at the set of open file descriptors inherited by the spawned process from the set of open file descriptors in the parent at the time of the *posix_spawn*( ) or *posix_spawnp*( ) call. It had been suggested that the *close*( ) and *dup2*( ) operations alone are sufficient to rearrange file descriptors, and that files which need to be opened for use by the spawned process can be handled either by having the calling process open them before the *posix_spawn*( ) or *posix_spawnp*( ) call (and close them after), or by passing filenames to the spawned process (in *argv*) so that it may open them itself. The standard developers recommend that applications use one of these two methods when practical, since detailed error status on a failed open operation is always available to the application this way. However, the standard developers feel that allowing a spawn file actions object to specify open operations is still appropriate because:

1. It is consistent with equivalent POSIX.5 (Ada) functionality.
2. It supports the I/O redirection paradigm commonly employed by POSIX programs designed to be invoked from a shell. When such a program is the child process, it may not be designed to open files on its own.
3. It allows file opens that might otherwise fail or violate file ownership/access rights if executed by the parent process.

Regarding 2. above, note that the spawn open file action provides to *posix_spawn*( ) and *posix_spawnp*( ) the same capability that the shell redirection operators provide to *system*( ), only without the intervening execution of a shell; for example:

```
system ("myprog <file1 3<file2");
```

Regarding 3. above, note that if the calling process needs to open one or more files for access by the spawned process, but has insufficient spare file descriptors, then the open action is necessary to allow the *open*( ) to occur in the context of the child process after other file descriptors have been closed (that must remain open in the parent).

Additionally, if a parent is executed from a file having a ‘‘set-user-id’’ mode bit set and the POSIX_SPAWN_RESETIDS flag is set in the spawn attributes, a file created within the parent process will (possibly incorrectly) have the parent’s effective user ID as its owner, whereas a file created via an *open*( ) action during *posix_spawn*( ) or *posix_spawnp*( ) will have the parent’s real ID as its owner; and an open by the parent process may successfully open a file to which the real user should not have access or fail to open a file to which the real user should have access.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

28853 **File Descriptor Mapping**

28854 The standard developers had originally proposed using an array which specified the mapping of
28855 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is
28856 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix_spawn*( )
28857 or *posix_spawnp*( ) without provision for one or more spare file descriptor entries (which simply
28858 may not be available). Such an array requires that an implementation develop a complex
28859 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor
28860 at the wrong time.

28861 It was noted by a member of the Ada Language Bindings working group that the approved Ada
28862 Language *Start_Process* family of POSIX process primitives use a caller-specified set of file
28863 actions to alter the normal *fork*( )/*exec* semantics for inheritance of file descriptors in a very
28864 flexible way, yet no such problems exist because the burden of determining how to achieve the
28865 final file descriptor mapping is completely on the application. Furthermore, although the file
28866 actions interface appears frightening at first glance, it is actually quite simple to implement in
28867 either a library or the kernel.

28868 **FUTURE DIRECTIONS**

28869 None.

28870 **SEE ALSO**

28871 *close*( ), *dup*( ), *open*( ), *posix_spawn*( ), *posix_spawn_file_actions_adddup2*( ),
28872 *posix_spawn_file_actions_destroy*( ), *posix_spawnp*( ), the Base Definitions volume of
28873 IEEE Std 1003.1-2001, **<spawn.h>**

28874 **CHANGE HISTORY**

28875 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28876 IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the
28877 string pointed to by *path* is copied by the *posix_spawn_file_actions_addopen*( ) function.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

28878 **NAME**

28879 posix_spawn_file_actions_adddup2 — add dup2 action to spawn file actions object
28880 (**ADVANCED REALTIME**)

28881 **SYNOPSIS**

28882 SPN
```
#include <spawn.h>

int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *
    file_actions, int fildes, int newfildes);
```
28883
28884
28885

28886 **DESCRIPTION**

28887 The *posix_spawn_file_actions_adddup2*( ) function shall add a *dup2*( ) action to the object
28888 referenced by *file_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as
28889 if *dup2*(*fildes*, *newfildes*) had been called) when a new process is spawned using this file actions
28890 object.

28891 A spawn file actions object is as defined in *posix_spawn_file_actions_addclose*( ).

28892 **RETURN VALUE**

28893 Upon successful completion, the *posix_spawn_file_actions_adddup2*( ) function shall return zero;
28894 otherwise, an error number shall be returned to indicate the error.

28895 **ERRORS**

28896 The *posix_spawn_file_actions_adddup2*( ) function shall fail if:

28897 [EBADF] The value specified by *fildes* or *newfildes* is negative or greater than or equal to
28898 {OPEN_MAX}.

28899 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28900 The *posix_spawn_file_actions_adddup2*( ) function may fail if:

28901 [EINVAL] The value specified by *file_actions* is invalid.

28902 It shall not be considered an error for the *fildes* argument passed to the
28903 *posix_spawn_file_actions_adddup2*( ) function to specify a file descriptor for which the specified
28904 operation could not be performed at the time of the call. Any such error will be detected when
28905 the associated file actions object is later used during a *posix_spawn*( ) or *posix_spawnp*( )
28906 operation.

28907 **EXAMPLES**

28908 None.

28909 **APPLICATION USAGE**

28910 The *posix_spawn_file_actions_adddup2*( ) function is part of the Spawn option and need not be
28911 provided on all implementations.

28912 **RATIONALE**

28913 Refer to the RATIONALE in *posix_spawn_file_actions_addclose*( ).

28914 **FUTURE DIRECTIONS**

28915 None.

28916 **SEE ALSO**

28917 *dup*( ), *posix_spawn*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_destroy*( ),
28918 *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

28919 **CHANGE HISTORY**

28920 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28921 IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the
28922 *newfildes* argument in addition to *fildes*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

28923 **NAME**

28924 posix_spawn_file_actions_addopen — add open action to spawn file actions object
28925 (**ADVANCED REALTIME**)

28926 **SYNOPSIS**

28927 SPN
```
#include <spawn.h>

int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *
    restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```
28931

28932 **DESCRIPTION**

28933 Refer to *posix_spawn_file_actions_addclose*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_spawn_file_actions_destroy, posix_spawn_file_actions_init — destroy and initialize spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

SPN

```
#include <spawn.h>

int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *
    file_actions);
int posix_spawn_file_actions_init(posix_spawn_file_actions_t *
    file_actions);
```

DESCRIPTION

The *posix_spawn_file_actions_destroy*( ) function shall destroy the object referenced by *file_actions*; the object becomes, in effect, uninitialized. An implementation may cause *posix_spawn_file_actions_destroy*( ) to set the object referenced by *file_actions* to an invalid value. A destroyed spawn file actions object can be reinitialized using *posix_spawn_file_actions_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

The *posix_spawn_file_actions_init*( ) function shall initialize the object referenced by *file_actions* to contain no file actions for *posix_spawn*( ) or *posix_spawnp*( ) to perform.

A spawn file actions object is as defined in *posix_spawn_file_actions_addclose*( ).

The effect of initializing an already initialized spawn file actions object is undefined.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

The *posix_spawn_file_actions_destroy*( ) function may fail if:

[EINVAL] The value specified by *file_actions* is invalid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

Refer to the RATIONALE in *posix_spawn_file_actions_addclose*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

28975 **NAME**

28976 posix_spawnattr_destroy, posix_spawnattr_init — destroy and initialize spawn attributes object
28977 (**ADVANCED REALTIME**)

28978 **SYNOPSIS**

28979 SPN
```
#include <spawn.h>

int posix_spawnattr_destroy(posix_spawnattr_t *attr);
int posix_spawnattr_init(posix_spawnattr_t *attr);
```
28982

28983 **DESCRIPTION**

28984 The *posix_spawnattr_destroy*( ) function shall destroy a spawn attributes object. A destroyed *attr*
28985 attributes object can be reinitialized using *posix_spawnattr_init*( ); the results of otherwise
28986 referencing the object after it has been destroyed are undefined. An implementation may cause
28987 *posix_spawnattr_destroy*( ) to set the object referenced by *attr* to an invalid value.

28988 The *posix_spawnattr_init*( ) function shall initialize a spawn attributes object *attr* with the default
28989 value for all of the individual attributes used by the implementation. Results are undefined if
28990 *posix_spawnattr_init*( ) is called specifying an already initialized *attr* attributes object.

28991 A spawn attributes object is of type **posix_spawnattr_t** (defined in **<spawn.h>**) and is used to
28992 specify the inheritance of process attributes across a spawn operation. IEEE Std 1003.1-2001 does
28993 not define comparison or assignment operators for the type **posix_spawnattr_t**.

28994 Each implementation shall document the individual attributes it uses and their default values
28995 unless these values are defined by IEEE Std 1003.1-2001. Attributes not defined by
28996 IEEE Std 1003.1-2001, their default values, and the names of the associated functions to get and
28997 set those attribute values are implementation-defined.

28998 The resulting spawn attributes object (possibly modified by setting individual attribute values),
28999 is used to modify the behavior of *posix_spawn*( ) or *posix_spawnp*( ). After a spawn attributes
29000 object has been used to spawn a process by a call to a *posix_spawn*( ) or *posix_spawnp*( ), any
29001 function affecting the attributes object (including destruction) shall not affect any process that
29002 has been spawned in this way.

29003 **RETURN VALUE**

29004 Upon successful completion, *posix_spawnattr_destroy*( ) and *posix_spawnattr_init*( ) shall return
29005 zero; otherwise, an error number shall be returned to indicate the error.

29006 **ERRORS**

29007 The *posix_spawnattr_init*( ) function shall fail if:

29008 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

29009 The *posix_spawnattr_destroy*( ) function may fail if:

29010 [EINVAL] The value specified by attr is invalid.

29011 **EXAMPLES**

29012 None.

29013 **APPLICATION USAGE**

29014 These functions are part of the Spawn option and need not be provided on all implementations.

29015 **RATIONALE**

29016 The original spawn interface proposed in IEEE Std 1003.1-2001 defined the attributes that specify
29017 the inheritance of process attributes across a spawn operation as a structure. In order to be able
29018 to separate optional individual attributes under their appropriate options (that is, the *spawn-*
29019 *schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

also for extensibility and consistency with the newer POSIX interfaces, the attributes interface has been changed to an opaque data type. This interface now consists of the type **posix_spawnattr_t**, representing a spawn attributes object, together with associated functions to initialize or destroy the attributes object, and to set or get each individual attribute. Although the new object-oriented interface is more verbose than the original structure, it is simple to use, more extensible, and easy to implement.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setflags*( ), *posix_spawnattr_setpgroup*( ), *posix_spawnattr_setsigmask*( ), *posix_spawnattr_setschedpolicy*( ), *posix_spawnattr_setschedparam*( ), *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already initialized spawn attributes option is undefined.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29039 **NAME**

29040 posix_spawnattr_getflags, posix_spawnattr_setflags — get and set the spawn-flags attribute of a
29041 spawn attributes object (**ADVANCED REALTIME**)

29042 **SYNOPSIS**

29043 SPN
```
#include <spawn.h>

int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
    short *restrict flags);
int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```
29044
29045
29046
29047

29048 **DESCRIPTION**

29049 The *posix_spawnattr_getflags*( ) function shall obtain the value of the *spawn-flags* attribute from
29050 the attributes object referenced by *attr*.

29051 The *posix_spawnattr_setflags*( ) function shall set the *spawn-flags* attribute in an initialized
29052 attributes object referenced by *attr*.

29053 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the
29054 new process image when invoking *posix_spawn*( ) or *posix_spawnp*( ). It is the bitwise-inclusive
29055 OR of zero or more of the following flags:

29056 POSIX_SPAWN_RESETIDS
29057 POSIX_SPAWN_SETPGROUP
29058 POSIX_SPAWN_SETSIGDEF
29059 POSIX_SPAWN_SETSIGMASK
29060 PS POSIX_SPAWN_SETSCHEDPARAM
29061 POSIX_SPAWN_SETSCHEDULER
29062

29063 These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags
29064 were set.

29065 **RETURN VALUE**

29066 Upon successful completion, *posix_spawnattr_getflags*( ) shall return zero and store the value of
29067 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an
29068 error number shall be returned to indicate the error.

29069 Upon successful completion, *posix_spawnattr_setflags*( ) shall return zero; otherwise, an error
29070 number shall be returned to indicate the error.

29071 **ERRORS**

29072 These functions may fail if:

29073 [EINVAL] The value specified by *attr* is invalid.

29074 The *posix_spawnattr_setflags*( ) function may fail if:

29075 [EINVAL] The value of the attribute being set is not valid.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29076 **EXAMPLES**

29077 None.

29078 **APPLICATION USAGE**

29079 These functions are part of the Spawn option and need not be provided on all implementations.

29080 **RATIONALE**

29081 None.

29082 **FUTURE DIRECTIONS**

29083 None.

29084 **SEE ALSO**

29085 *posix_spawn*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_init*( ), *posix_spawnattr_getsigdefault*( ),
29086 *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ),
29087 *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setpgroup*( ),
29088 *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setschedpolicy*( ), *posix_spawnattr_setsigmask*( ),
29089 *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

29090 **CHANGE HISTORY**

29091 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29092 **NAME**

29093 posix_spawnattr_getpgroup, posix_spawnattr_setpgroup — get and set the spawn-pgroup
29094 attribute of a spawn attributes object (**ADVANCED REALTIME**)

29095 **SYNOPSIS**

29096 SPN
```
#include <spawn.h>

int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
    pid_t *restrict pgroup);
int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```
29100

29101 **DESCRIPTION**

29102 The *posix_spawnattr_getpgroup*( ) function shall obtain the value of the *spawn-pgroup* attribute
29103 from the attributes object referenced by *attr*.

29104 The *posix_spawnattr_setpgroup*( ) function shall set the *spawn-pgroup* attribute in an initialized
29105 attributes object referenced by *attr*.

29106 The *spawn-pgroup* attribute represents the process group to be joined by the new process image
29107 in a spawn operation (if POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute). The
29108 default value of this attribute shall be zero.

29109 **RETURN VALUE**

29110 Upon successful completion, *posix_spawnattr_getpgroup*( ) shall return zero and store the value of
29111 the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise,
29112 an error number shall be returned to indicate the error.

29113 Upon successful completion, *posix_spawnattr_setpgroup*( ) shall return zero; otherwise, an error
29114 number shall be returned to indicate the error.

29115 **ERRORS**

29116 These functions may fail if:

29117 [EINVAL] The value specified by *attr* is invalid.

29118 The *posix_spawnattr_setpgroup*( ) function may fail if:

29119 [EINVAL] The value of the attribute being set is not valid.

29120 **EXAMPLES**

29121 None.

29122 **APPLICATION USAGE**

29123 These functions are part of the Spawn option and need not be provided on all implementations.

29124 **RATIONALE**

29125 None.

29126 **FUTURE DIRECTIONS**

29127 None.

29128 **SEE ALSO**

29129 *posix_spawn*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_init*( ), *posix_spawnattr_getsigdefault*( ),
29130 *posix_spawnattr_getflags*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ),
29131 *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setflags*( ),
29132 *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setschedpolicy*( ), *posix_spawnattr_setsigmask*( ),
29133 *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29134 **CHANGE HISTORY**

29135 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29136 **NAME**

29137 posix_spawnattr_getschedparam, posix_spawnattr_setschedparam — get and set the spawn-
29138 schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

29139 **SYNOPSIS**

29140 SPN PS
```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedparam(const posix_spawnattr_t *
    restrict attr, struct sched_param *restrict schedparam);
int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
    const struct sched_param *restrict schedparam);
```
29146

29147 **DESCRIPTION**

29148 The *posix_spawnattr_getschedparam*( ) function shall obtain the value of the *spawn-schedparam*
29149 attribute from the attributes object referenced by *attr*.

29150 The *posix_spawnattr_setschedparam*( ) function shall set the *spawn-schedparam* attribute in an
29151 initialized attributes object referenced by *attr*.

29152 The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new
29153 process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER or
29154 POSIX_SPAWN_SETSCHEDPARAM is set in the *spawn-flags* attribute). The default value of this
29155 attribute is unspecified.

29156 **RETURN VALUE**

29157 Upon successful completion, *posix_spawnattr_getschedparam*( ) shall return zero and store the
29158 value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam*
29159 parameter; otherwise, an error number shall be returned to indicate the error.

29160 Upon successful completion, *posix_spawnattr_setschedparam*( ) shall return zero; otherwise, an
29161 error number shall be returned to indicate the error.

29162 **ERRORS**

29163 These functions may fail if:

29164 [EINVAL] The value specified by *attr* is invalid.

29165 The *posix_spawnattr_setschedparam*( ) function may fail if:

29166 [EINVAL] The value of the attribute being set is not valid.

29167 **EXAMPLES**

29168 None.

29169 **APPLICATION USAGE**

29170 These functions are part of the Spawn and Process Scheduling options and need not be provided
29171 on all implementations.

29172 **RATIONALE**

29173 None.

29174 **FUTURE DIRECTIONS**

29175 None.

29176 **SEE ALSO**

29177 *posix_spawn*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_init*( ), *posix_spawnattr_getsigdefault*( ),
29178 *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedpolicy*( ),
29179 *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setflags*( ),

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29180 *posix_spawnattr_setpgroup*( ), *posix_spawnattr_setschedpolicy*( ), *posix_spawnattr_setsigmask*( ),
29181 *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**, **<spawn.h>**

29182 **CHANGE HISTORY**

29183 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy — get and set the spawn-schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

SPN PS
```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *
    restrict attr, int *restrict schedpolicy);
int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
    int schedpolicy);
```

DESCRIPTION

The *posix_spawnattr_getschedpolicy*( ) function shall obtain the value of the *spawn-schedpolicy* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setschedpolicy*( ) function shall set the *spawn-schedpolicy* attribute in an initialized attributes object referenced by *attr*.

The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER is set in the *spawn-flags* attribute). The default value of this attribute is unspecified.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getschedpolicy*( ) shall return zero and store the value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setschedpolicy*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *posix_spawnattr_setschedpolicy*( ) function may fail if:

[EINVAL] The value of the attribute being set is not valid.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn and Process Scheduling options and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_init*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setflags*( ), *posix_spawnattr_setpgroup*( ), *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setsigmask*( ),

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

posix_spawnp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**, **<spawn.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29231 **NAME**

29232 posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault — get and set the spawn-
29233 sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

29234 **SYNOPSIS**

29235 SPN
```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigdefault(const posix_spawnattr_t *
    restrict attr, sigset_t *restrict sigdefault);
int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigdefault);
```
29241

29242 **DESCRIPTION**

29243 The *posix_spawnattr_getsigdefault*( ) function shall obtain the value of the *spawn-sigdefault*
29244 attribute from the attributes object referenced by *attr*.

29245 The *posix_spawnattr_setsigdefault*( ) function shall set the *spawn-sigdefault* attribute in an
29246 initialized attributes object referenced by *attr*.

29247 The *spawn-sigdefault* attribute represents the set of signals to be forced to default signal handling
29248 in the new process image (if POSIX_SPAWN_SETSIGDEF is set in the *spawn-flags* attribute) by a
29249 spawn operation. The default value of this attribute shall be an empty signal set.

29250 **RETURN VALUE**

29251 Upon successful completion, *posix_spawnattr_getsigdefault*( ) shall return zero and store the value
29252 of the *spawn-sigdefault* attribute of *attr* into the object referenced by the *sigdefault* parameter;
29253 otherwise, an error number shall be returned to indicate the error.

29254 Upon successful completion, *posix_spawnattr_setsigdefault*( ) shall return zero; otherwise, an error
29255 number shall be returned to indicate the error.

29256 **ERRORS**

29257 These functions may fail if:

29258 [EINVAL] The value specified by *attr* is invalid.

29259 The *posix_spawnattr_setsigdefault*( ) function may fail if:

29260 [EINVAL] The value of the attribute being set is not valid.

29261 **EXAMPLES**

29262 None.

29263 **APPLICATION USAGE**

29264 These functions are part of the Spawn option and need not be provided on all implementations.

29265 **RATIONALE**

29266 None.

29267 **FUTURE DIRECTIONS**

29268 None.

29269 **SEE ALSO**

29270 *posix_spawn*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_init*( ), *posix_spawnattr_getflags*( ),
29271 *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ),
29272 *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setflags*( ), *posix_spawnattr_setpgroup*( ),
29273 *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setschedpolicy*( ), *posix_spawnattr_setsigmask*( ),
29274 *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<spawn.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29275 **CHANGE HISTORY**

29276 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29277 **NAME**

29278 posix_spawnattr_getsigmask, posix_spawnattr_setsigmask — get and set the spawn-sigmask
29279 attribute of a spawn attributes object (**ADVANCED REALTIME**)

29280 **SYNOPSIS**

29281 SPN
```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
    sigset_t *restrict sigmask);
int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```
29287

29288 **DESCRIPTION**

29289 The *posix_spawnattr_getsigmask*( ) function shall obtain the value of the *spawn-sigmask* attribute
29290 from the attributes object referenced by *attr*.

29291 The *posix_spawnattr_setsigmask*( ) function shall set the *spawn-sigmask* attribute in an initialized
29292 attributes object referenced by *attr*.

29293 The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a
29294 spawn operation (if POSIX_SPAWN_SETSIGMASK is set in the *spawn-flags* attribute). The
29295 default value of this attribute is unspecified.

29296 **RETURN VALUE**

29297 Upon successful completion, *posix_spawnattr_getsigmask*( ) shall return zero and store the value
29298 of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter;
29299 otherwise, an error number shall be returned to indicate the error.

29300 Upon successful completion, *posix_spawnattr_setsigmask*( ) shall return zero; otherwise, an error
29301 number shall be returned to indicate the error.

29302 **ERRORS**

29303 These functions may fail if:

29304 [EINVAL] The value specified by *attr* is invalid.

29305 The *posix_spawnattr_setsigmask*( ) function may fail if:

29306 [EINVAL] The value of the attribute being set is not valid.

29307 **EXAMPLES**

29308 None.

29309 **APPLICATION USAGE**

29310 These functions are part of the Spawn option and need not be provided on all implementations.

29311 **RATIONALE**

29312 None.

29313 **FUTURE DIRECTIONS**

29314 None.

29315 **SEE ALSO**

29316 *posix_spawn*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_init*( ), *posix_spawnattr_getsigdefault*( ),
29317 *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ),
29318 *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setflags*( ),
29319 *posix_spawnattr_setpgroup*( ), *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setschedpolicy*( ),
29320 *posix_spawnp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<spawn.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29321 **CHANGE HISTORY**

29322 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29323 **NAME**

29324 posix_spawnattr_init — initialize the spawn attributes object (**ADVANCED REALTIME**)

29325 **SYNOPSIS**

29326 SPN
```
#include <spawn.h>
```

29327
```
int posix_spawnattr_init(posix_spawnattr_t *attr);
```

29328

29329 **DESCRIPTION**

29330 Refer to *posix_spawnattr_destroy*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29331 **NAME**

29332 posix_spawnattr_setflags — set the spawn-flags attribute of a spawn attributes object
29333 (**ADVANCED REALTIME**)

29334 **SYNOPSIS**

29335 SPN
```
#include <spawn.h>
```

29336
```
int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

29337

29338 **DESCRIPTION**

29339 Refer to *posix_spawnattr_getflags*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29340 **NAME**

29341 posix_spawnattr_setpgroup — set the spawn-pgroup attribute of a spawn attributes object
29342 (**ADVANCED REALTIME**)

29343 **SYNOPSIS**

29344 SPN
```
#include <spawn.h>
```

29345
```
int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

29346

29347 **DESCRIPTION**

29348 Refer to *posix_spawnattr_getpgroup*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29349 **NAME**

29350 posix_spawnattr_setschedparam — set the spawn-schedparam attribute of a spawn attributes
29351 object (**ADVANCED REALTIME**)

29352 **SYNOPSIS**

29353 SPN PS
```
#include <sched.h>
#include <spawn.h>

int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
    const struct sched_param *restrict schedparam);
```
29354
29355
29356
29357

29358 **DESCRIPTION**

29359 Refer to *posix_spawnattr_getschedparam*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29360 **NAME**

29361 posix_spawnattr_setschedpolicy — set the spawn-schedpolicy attribute of a spawn attributes
29362 object (**ADVANCED REALTIME**)

29363 **SYNOPSIS**

29364 SPN PS
```
#include <sched.h>
#include <spawn.h>

int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
    int schedpolicy);
```
29365
29366
29367
29368

29369 **DESCRIPTION**

29370 Refer to *posix_spawnattr_getschedpolicy*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

29371 **NAME**

29372 posix_spawnattr_setsigdefault — set the spawn-sigdefault attribute of a spawn attributes object
29373 (**ADVANCED REALTIME**)

29374 **SYNOPSIS**

29375 SPN
```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigdefault);
```
29376
29377
29378
29379

29380 **DESCRIPTION**

29381 Refer to *posix_spawnattr_getsigdefault*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29382 **NAME**

29383 posix_spawnattr_setsigmask — set the spawn-sigmask attribute of a spawn attributes object
29384 (**ADVANCED REALTIME**)

29385 **SYNOPSIS**

29386 SPN
```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```
29390

29391 **DESCRIPTION**

29392 Refer to *posix_spawnattr_getsigmask*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

29393 **NAME**

29394 posix_spawnp — spawn a process (**ADVANCED REALTIME**)

29395 **SYNOPSIS**

29396 SPN
```
#include <spawn.h>
```

29397
```
int posix_spawnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
```

29401

29402 **DESCRIPTION**

29403 Refer to *posix_spawn*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29404 **NAME**

29405 posix_trace_attr_destroy, posix_trace_attr_init — destroy and initialize the trace stream
29406 attributes object (**TRACING**)

29407 **SYNOPSIS**

29408 TRC
```
#include <trace.h>

int posix_trace_attr_destroy(trace_attr_t *attr);
int posix_trace_attr_init(trace_attr_t *attr);
```
29411

29412 **DESCRIPTION**

29413 The *posix_trace_attr_destroy*( ) function shall destroy an initialized trace attributes object. A
29414 destroyed *attr* attributes object can be reinitialized using *posix_trace_attr_init*( ); the results of
29415 otherwise referencing the object after it has been destroyed are undefined.

29416 The *posix_trace_attr_init*( ) function shall initialize a trace attributes object *attr* with the default
29417 value for all of the individual attributes used by a given implementation. The read-only
29418 *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object
29419 shall be set to their appropriate values (see Section 2.11.1.2 (on page 75)).

29420 Results are undefined if *posix_trace_attr_init*( ) is called specifying an already initialized *attr*
29421 attributes object.

29422 Implementations may add extensions to the trace attributes object structure as permitted in the
29423 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance.

29424 The resulting attributes object (possibly modified by setting individual attributes values), when
29425 used by *posix_trace_create*( ), defines the attributes of the trace stream created. A single attributes
29426 object can be used in multiple calls to *posix_trace_create*( ). After one or more trace streams have
29427 been created using an attributes object, any function affecting that attributes object, including
29428 destruction, shall not affect any trace stream previously created. An initialized attributes object
29429 also serves to receive the attributes of an existing trace stream or trace log when calling the
29430 *posix_trace_get_attr*( ) function.

29431 **RETURN VALUE**

29432 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
29433 return the corresponding error number.

29434 **ERRORS**

29435 The *posix_trace_attr_destroy*( ) function may fail if:

29436 [EINVAL] The value of *attr* is invalid.

29437 The *posix_trace_attr_init*( ) function shall fail if:

29438 [ENOMEM] Insufficient memory exists to initialize the trace attributes object.

29439 **EXAMPLES**

29440 None.

29441 **APPLICATION USAGE**

29442 None.

29443 **RATIONALE**

29444 None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29445 **FUTURE DIRECTIONS**

29446 None.

29447 **SEE ALSO**

29448 *posix_trace_create*( ), *posix_trace_get_attr*( ), *uname*( ), the Base Definitions volume of
29449 IEEE Std 1003.1-2001, **<trace.h>**

29450 **CHANGE HISTORY**

29451 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29452 IEEE PASC Interpretation 1003.1 #123 is applied.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_trace_attr_getclockres, posix_trace_attr_getcreatetime, posix_trace_attr_getgenversion, posix_trace_attr_getname, posix_trace_attr_setname — retrieve and set information about a trace stream (**TRACING**)

SYNOPSIS

TRC
```
#include <time.h>
#include <trace.h>

int posix_trace_attr_getclockres(const trace_attr_t *attr,
    struct timespec *resolution);
int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
    struct timespec *createtime);

#include <trace.h>

int posix_trace_attr_getgenversion(const trace_attr_t *attr,
    char *genversion);
int posix_trace_attr_getname(const trace_attr_t *attr,
    char *tracename);
int posix_trace_attr_setname(trace_attr_t *attr,
    const char *tracename);
```

DESCRIPTION

The *posix_trace_attr_getclockres*( ) function shall copy the clock resolution of the clock used to generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the *attr* argument into the structure pointed to by the *resolution* argument.

The *posix_trace_attr_getcreatetime*( ) function shall copy the trace stream creation time from the *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of creation of the trace stream.

The *posix_trace_attr_getgenversion*( ) function shall copy the string containing version information from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of a character array which can store at least {TRACE_NAME_MAX} characters.

The *posix_trace_attr_getname*( ) function shall copy the string containing the trace name from the *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character array which can store at least {TRACE_NAME_MAX} characters.

The *posix_trace_attr_setname*( ) function shall set the name in the *trace-name* attribute of the attributes object pointed to by the *attr* argument, using the trace name string supplied by the *tracename* argument. If the supplied string contains more than {TRACE_NAME_MAX} characters, the name copied into the *trace-name* attribute may be truncated to one less than the length of {TRACE_NAME_MAX} characters. The default value is a null string.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, the *posix_trace_attr_getclockres*( ) function stores the *clock-resolution* attribute value in the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If successful, the *posix_trace_attr_getcreatetime*( ) function stores the trace stream creation time in the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

If successful, the *posix_trace_attr_getgenversion*( ) function stores the trace version information in the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

If successful, the *posix_trace_attr_getname*( ) function stores the trace name in the string pointed to by *tracename*. Otherwise, the content of this string is unspecified.

ERRORS

The *posix_trace_attr_getclockres*( ), *posix_trace_attr_getcreatetime*( ), *posix_trace_attr_getgenversion*( ), and *posix_trace_attr_getname*( ) functions may fail if:

[EINVAL] The value specified by one of the arguments is invalid.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_trace_attr_init( ), *posix_trace_create*( ), *posix_trace_get_attr*( ), *uname*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**, **<trace.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy, posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

SYNOPSIS

TRC
```
#include <trace.h>
```

TRC TRI
```
int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
    int *restrict inheritancepolicy);
```
TRC TRL
```
int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
    int *restrict logpolicy);
```
TRC
```
int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
    int *streampolicy);
```
TRC TRI
```
int posix_trace_attr_setinherited(trace_attr_t *attr,
    int inheritancepolicy);
```
TRC TRL
```
int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
    int logpolicy);
```
TRC
```
int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
    int streampolicy);
```

DESCRIPTION

TRI The *posix_trace_attr_getinherited*( ) and *posix_trace_attr_setinherited*( ) functions, respectively, shall get and set the inheritance policy stored in the *inheritance* attribute for traced processes across the *fork*( ) and *spawn*( ) operations. The *inheritance* attribute of the attributes object pointed to by the *attr* argument shall be set to one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_CLOSE_FOR_CHILD
: After a *fork*( ) or *spawn*( ) operation, the child shall not be traced, and tracing of the parent shall continue.

POSIX_TRACE_INHERITED
: After a *fork*( ) or *spawn*( ) operation, if the parent is being traced, its child shall be concurrently traced using the same trace stream.

The default value for the *inheritance* attribute is POSIX_TRACE_CLOSE_FOR_CHILD.

TRL The *posix_trace_attr_getlogfullpolicy*( ) and *posix_trace_attr_setlogfullpolicy*( ) functions, respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the attributes object pointed to by the *attr* argument.

The *log-full-policy* attribute shall be set to one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_LOOP
: The trace log shall loop until the associated trace stream is stopped. This policy means that when the trace log gets full, the file system shall reuse the resources allocated to the oldest trace events that were recorded. In this way, the trace log will always contain the most recent trace events flushed.

POSIX_TRACE_UNTIL_FULL
: The trace stream shall be flushed to the trace log until the trace log is full. This condition can be deduced from the *posix_log_full_status* member status (see the **posix_trace_status_info** structure defined in **<trace.h>**). The last recorded trace event shall be the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29568 POSIX_TRACE_STOP trace event.

29569 POSIX_TRACE_APPEND
29570 The associated trace stream shall be flushed to the trace log without log size limitation. If
29571 the application specifies POSIX_TRACE_APPEND, the implementation shall ignore the
29572 *log-max-size* attribute.

29573 The default value for the *log-full-policy* attribute is POSIX_TRACE_LOOP.

29574 The *posix_trace_attr_getstreamfullpolicy*( ) and *posix_trace_attr_setstreamfullpolicy*( ) functions,
29575 respectively, shall get and set the trace stream full policy stored in the *stream-full-policy* attribute
29576 of the attributes object pointed to by the *attr* argument.

29577 The *stream-full-policy* attribute shall be set to one of the following values defined by manifest
29578 constants in the **<trace.h>** header:

29579 POSIX_TRACE_LOOP
29580 The trace stream shall loop until explicitly stopped by the *posix_trace_stop*( ) function. This
29581 policy means that when the trace stream is full, the trace system shall reuse the resources
29582 allocated to the oldest trace events recorded. In this way, the trace stream will always
29583 contain the most recent trace events recorded.

29584 POSIX_TRACE_UNTIL_FULL
29585 The trace stream will run until the trace stream resources are exhausted. Then the trace
29586 stream will stop. This condition can be deduced from *posix_stream_status* and
29587 *posix_stream_full_status* (see the **posix_trace_status_info** structure defined in **<trace.h>**).
29588 When this trace stream is read, a POSIX_TRACE_STOP trace event shall be reported after
29589 reporting the last recorded trace event. The trace system shall reuse the resources allocated
29590 to any trace events already reported—see the *posix_trace_getnext_event*( ),
29591 *posix_trace_trygetnext_event*( ), and *posix_trace_timedgetnext_event*( ) functions—or already
29592 flushed for an active trace stream with log if the Trace Log option is supported; see the
29593 *posix_trace_flush*( ) function. The trace system shall restart the trace stream when it is empty
29594 and may restart it sooner. A POSIX_TRACE_START trace event shall be reported before
29595 reporting the next recorded trace event.

29596 TRL POSIX_TRACE_FLUSH
29597 If the Trace Log option is supported, this policy is identical to the
29598 POSIX_TRACE_UNTIL_FULL trace stream full policy except that the trace stream shall be
29599 flushed regularly as if *posix_trace_flush*( ) had been explicitly called. Defining this policy for
29600 an active trace stream without log shall be invalid.

29601 The default value for the *stream-full-policy* attribute shall be POSIX_TRACE_LOOP for an active
29602 trace stream without log.

29603 TRL If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be
29604 POSIX_TRACE_FLUSH for an active trace stream with log.

29605 **RETURN VALUE**

29606 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
29607 return the corresponding error number.

29608 TRI If successful, the *posix_trace_attr_getinherited*( ) function shall store the *inheritance* attribute value
29609 in the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.

29610 TRL If successful, the *posix_trace_attr_getlogfullpolicy*( ) function shall store the *log-full-policy* attribute
29611 value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.

29612 If successful, the *posix_trace_attr_getstreamfullpolicy*( ) function shall store the *stream-full-policy*
29613 attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

undefined.

ERRORS

These functions may fail if:

[EINVAL] The value specified by at least one of the arguments is invalid.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fork( ), *posix_trace_attr_init*( ), *posix_trace_create*( ), *posix_trace_flush*( ), *posix_trace_get_attr*( ), *posix_trace_getnext_event*( ), *posix_trace_start*( ), *posix_trace_timedgetnext_event*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<trace.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/39 is applied, adding the TRL and TRC margin codes to the *posix_trace_attr_setlogfullpolicy*( ) function.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize, posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxuserevent size, posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize, posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

SYNOPSIS

TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
    size_t *restrict logsize);
int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
    size_t *restrict maxdatasize);
int posix_trace_attr_getmaxsystemeventsize(
    const trace_attr_t *restrict attr,
    size_t *restrict eventsize);
int posix_trace_attr_getmaxuserevent size(
    const trace_attr_t *restrict attr,
    size_t data_len, size_t *restrict eventsize);
int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
    size_t *restrict streamsize);
int posix_trace_attr_setlogsize(trace_attr_t *attr,
    size_t logsize);
int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
    size_t maxdatasize);
int posix_trace_attr_setstreamsize(trace_attr_t *attr,
    size_t streamsize);
```

DESCRIPTION

TRL The *posix_trace_attr_getlogsize*( ) function shall copy the log size, in bytes, from the *log-max-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for system and user trace events in the trace log. The default value for the *log-max-size* attribute is implementation-defined.

The *posix_trace_attr_setlogsize*( ) function shall set the maximum allowed size, in bytes, in the *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *logsize* argument.

The trace log size shall be used if the *log-full-policy* attribute is set to POSIX_TRACE_LOOP or POSIX_TRACE_UNTIL_FULL. If the *log-full-policy* attribute is set to POSIX_TRACE_APPEND, the implementation shall ignore the *log-max-size* attribute.

The *posix_trace_attr_getmaxdatasize*( ) function shall copy the maximum user trace event data size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *maxdatasize* argument. The default value for the *max-data-size* attribute is implementation-defined.

The *posix_trace_attr_getmaxsystemeventsize*( ) function shall calculate the maximum memory size, in bytes, required to store a single system trace event. This value is calculated for the trace stream attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the *eventsize* argument.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The values returned as the maximum memory sizes of the user and system trace events shall be such that if the sum of the maximum memory sizes of a set of the trace events that may be recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace stream, the system provides the necessary resources for recording all those trace events, without loss.

The *posix_trace_attr_getmaxusereventsize*( ) function shall calculate the maximum memory size, in bytes, required to store a single user trace event generated by a call to *posix_trace_event*( ) with a *data_len* parameter equal to the *data_len* value specified in this call. This value is calculated for the trace stream attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the *eventsize* argument.

The *posix_trace_attr_getstreamsize*( ) function shall copy the stream size, in bytes, from the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *streamsize* argument.

This stream size is the current total memory size reserved for system and user trace events in the trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The stream size refers to memory used to store trace event records. Other stream data (for example, trace attribute values) shall not be included in this size.

The *posix_trace_attr_setmaxdatasize*( ) function shall set the maximum allowed size, in bytes, in the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size for the user data argument which may be passed to *posix_trace_event*( ). The implementation shall be allowed to truncate data passed to *trace_user_event* which is longer than *maxdatasize*.

The *posix_trace_attr_setstreamsize*( ) function shall set the minimum allowed size, in bytes, in the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *streamsize* argument.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

TRL The *posix_trace_attr_getlogsize*( ) function stores the maximum trace log allowed size in the object pointed to by *logsize*, if successful.

The *posix_trace_attr_getmaxdatasize*( ) function stores the maximum trace event record memory size in the object pointed to by *maxdatasize*, if successful.

The *posix_trace_attr_getmaxsystemeventsize*( ) function stores the maximum memory size to store a single system trace event in the object pointed to by *eventsize*, if successful.

The *posix_trace_attr_getmaxusereventsize*( ) function stores the maximum memory size to store a single user trace event in the object pointed to by *eventsize*, if successful.

The *posix_trace_attr_getstreamsize*( ) function stores the maximum trace stream allowed size in the object pointed to by *streamsize*, if successful.

ERRORS

These functions may fail if:

[EINVAL] The value specified by one of the arguments is invalid.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_trace_attr_init( ), *posix_trace_create*( ), *posix_trace_event*( ), *posix_trace_get_attr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<trace.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29735 **NAME**

29736 posix_trace_attr_getname — retrieve and set information about a trace stream (**TRACING**)

29737 **SYNOPSIS**

29738 TRC
```
#include <trace.h>
```

29739
```
int posix_trace_attr_getname(const trace_attr_t *attr,
    char *tracename);
```

29741

29742 **DESCRIPTION**

29743 Refer to *posix_trace_attr_getclockres*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29744 **NAME**

29745 posix_trace_attr_getstreamfullpolicy — retrieve and set the behavior of a trace stream
29746 (**TRACING**)

29747 **SYNOPSIS**

29748 TRC
```
#include <trace.h>
```

29749
```
int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
    int *streampolicy);
```

29751

29752 **DESCRIPTION**

29753 Refer to *posix_trace_attr_getinherited*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29754 **NAME**

29755 posix_trace_attr_getstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29756 **SYNOPSIS**

29757 TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
    size_t *restrict streamsize);
```
29761

29762 **DESCRIPTION**

29763 Refer to *posix_trace_attr_getlogsize*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29764 **NAME**

29765 posix_trace_attr_init — initialize the trace stream attributes object (**TRACING**)

29766 **SYNOPSIS**

29767 TRC
```
#include <trace.h>
```

29768
```
int posix_trace_attr_init(trace_attr_t *attr);
```

29769

29770 **DESCRIPTION**

29771 Refer to *posix_trace_attr_destroy*( ).

-·-`,,,`·-·-`,,`,,`,`,,`·---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29772 **NAME**

29773 posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy — retrieve and set the behavior
29774 of a trace stream (**TRACING**)

29775 **SYNOPSIS**

29776 TRC
```
#include <trace.h>
```

29777 TRC TRI
```
int posix_trace_attr_setinherited(trace_attr_t *attr,
    int inheritancepolicy);
```

29779 TRC TRL
```
int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
    int logpolicy);
```

29781

29782 **DESCRIPTION**

29783 Refer to *posix_trace_attr_getinherited*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

29784 **NAME**

29785 posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize — retrieve and set trace stream
29786 size attributes (**TRACING**)

29787 **SYNOPSIS**

29788 TRC
```
#include <sys/types.h>
#include <trace.h>
```

29790 TRC TRL
```
int posix_trace_attr_setlogsize(trace_attr_t *attr,
    size_t logsize);
```

29792 TRC
```
int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
    size_t maxdatasize);
```

29794

29795 **DESCRIPTION**

29796 Refer to *posix_trace_attr_getlogsize*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29797 **NAME**

29798 posix_trace_attr_setname — retrieve and set information about a trace stream (**TRACING**)

29799 **SYNOPSIS**

29800 TRC
```
#include <trace.h>

int posix_trace_attr_setname(trace_attr_t *attr,
    const char *tracename);
```
29801

29802

29803

29804 **DESCRIPTION**

29805 Refer to *posix_trace_attr_getclockres*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_trace_attr_setstreamfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

SYNOPSIS

TRC
```
#include <trace.h>

int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
    int streampolicy);
```

DESCRIPTION

Refer to *posix_trace_attr_getinherited*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

29816 **NAME**

29817 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29818 **SYNOPSIS**

29819 TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_setstreamsize(trace_attr_t *attr,
    size_t streamsize);
```
29823

29824 **DESCRIPTION**

29825 Refer to *posix_trace_attr_getlogsize*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

posix_trace_clear — clear trace stream and trace log (**TRACING**)

SYNOPSIS

TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_clear(trace_id_t trid);
```

DESCRIPTION

The *posix_trace_clear*( ) function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create*( ) function, except that the same allocated resources shall be reused, the mapping of trace event type identifiers to trace event names shall be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

All trace events in the trace stream recorded before the call to *posix_trace_clear*( ) shall be lost. The *posix_stream_full_status* status shall be set to POSIX_TRACE_NOT_FULL. There is no guarantee that all trace events that occurred during the *posix_trace_clear*( ) call are recorded; the behavior with respect to trace points that may occur during this call is unspecified.

TRL If the Trace Log option is supported and the trace stream has been created with a log, the *posix_trace_clear*( ) function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create_withlog*( ) function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix_trace_clear*( ) shall be the same as the first trace event recorded in the active trace stream after the call to *posix_trace_clear*( ). The *posix_log_full_status* status shall be set to POSIX_TRACE_NOT_FULL. There is no guarantee that all trace events that occurred during the *posix_trace_clear*( ) call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is POSIX_TRACE_APPEND, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

RETURN VALUE

Upon successful completion, the *posix_trace_clear*( ) function shall return a value of zero. Otherwise, it shall return the corresponding error number.

ERRORS

The *posix_trace_clear*( ) function shall fail if:

[EINVAL] The value of the *trid* argument does not correspond to an active trace stream.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29870 **SEE ALSO**

29871 *posix_trace_attr_init*( ), *posix_trace_create*( ), *posix_trace_flush*( ), *posix_trace_get_attr*( ), the Base
29872 Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<trace.h>**

29873 **CHANGE HISTORY**

29874 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29875 IEEE PASC Interpretation 1003.1 #123 is applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

posix_trace_close, posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)

SYNOPSIS

TRC TRL
```
#include <trace.h>

int posix_trace_close(trace_id_t trid);
int posix_trace_open(int file_desc, trace_id_t *trid);
int posix_trace_rewind(trace_id_t trid);
```

DESCRIPTION

The *posix_trace_close*( ) function shall deallocate the trace log identifier indicated by *trid*, and all of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall fail.

The *posix_trace_open*( ) function shall allocate the necessary resources and establish the connection between a trace log identified by the *file_desc* argument and a trace stream identifier identified by the object pointed to by the *trid* argument. The *file_desc* argument should be a valid open file descriptor that corresponds to a trace log. The *file_desc* argument shall be open for reading. The current trace event timestamp, which specifies the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event*( ), shall be set to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

The *posix_trace_open*( ) function shall return a trace stream identifier in the variable pointed to by the *trid* argument, that may only be used by the following functions:

posix_trace_close( )	*posix_trace_get_attr*( )
posix_trace_eventid_equal( )	*posix_trace_get_status*( )
posix_trace_eventid_get_name( )	*posix_trace_getnext_event*( )
posix_trace_eventtypelist_getnext_id( )	*posix_trace_rewind*( )
posix_trace_eventtypelist_rewind( )	

In particular, notice that the operations normally used by a trace controller process, such as *posix_trace_start*( ), *posix_trace_stop*( ), or *posix_trace_shutdown*( ), cannot be invoked using the trace stream identifier returned by the *posix_trace_open*( ) function.

The *posix_trace_rewind*( ) function shall reset the current trace event timestamp, which specifies the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event*( ), to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, the *posix_trace_open*( ) function stores the trace stream identifier value in the object pointed to by *trid*.

ERRORS

The *posix_trace_open*( ) function shall fail if:

[EINTR] The operation was interrupted by a signal and thus no trace log was opened.

[EINVAL] The object pointed to by *file_desc* does not correspond to a valid trace log.

The *posix_trace_close*( ) and *posix_trace_rewind*( ) functions may fail if:

[EINVAL] The object pointed to by *trid* does not correspond to a valid trace log.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29919 **EXAMPLES**

29920 None.

29921 **APPLICATION USAGE**

29922 None.

29923 **RATIONALE**

29924 None.

29925 **FUTURE DIRECTIONS**

29926 None.

29927 **SEE ALSO**

29928 *posix_trace_get_attr*( ), *posix_trace_get_filter*( ), *posix_trace_getnext_event*( ), the Base Definitions
29929 volume of IEEE Std 1003.1-2001, **<trace.h>**

29930 **CHANGE HISTORY**

29931 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29932 IEEE PASC Interpretation 1003.1 #123 is applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

posix_trace_create, posix_trace_create_withlog, posix_trace_flush, posix_trace_shutdown — trace stream initialization, flush, and shutdown from a process (**TRACING**)

SYNOPSIS

TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_create(pid_t pid,
    const trace_attr_t *restrict attr,
    trace_id_t *restrict trid);
```
TRC TRL
```
int posix_trace_create_withlog(pid_t pid,
    const trace_attr_t *restrict attr, int file_desc,
    trace_id_t *restrict trid);
int posix_trace_flush(trace_id_t trid);
```
TRC
```
int posix_trace_shutdown(trace_id_t trid);
```

DESCRIPTION

The *posix_trace_create*( ) function shall create an active trace stream. It allocates all the resources needed by the trace stream being created for tracing the process specified by *pid* in accordance with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and shall have been initialized by the function *posix_trace_attr_init*( ) prior to the *posix_trace_create*( ) call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object shall be manipulated through a set of functions described in the *posix_trace_attr* family of functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream shall be set to the value of the system clock, if the Timers option is not supported, or to the value of the CLOCK_REALTIME clock, if the Timers option is supported.

The *pid* argument represents the target process to be traced. If the process executing this function does not have appropriate privileges to trace the process identified by *pid*, an error shall be returned. If the *pid* argument is zero, the calling process shall be traced.

The *posix_trace_create*( ) function shall store the trace stream identifier of the new trace stream in the object pointed to by the *trid* argument. This trace stream identifier shall be used in subsequent calls to control tracing. The *trid* argument may only be used by the following functions:

posix_trace_clear( )	*posix_trace_getnext_event*( )
posix_trace_eventid_equal( )	*posix_trace_shutdown*( )
posix_trace_eventid_get_name( )	*posix_trace_start*( )
posix_trace_eventtypelist_getnext_id( )	*posix_trace_stop*( )
posix_trace_eventtypelist_rewind( )	*posix_trace_timedgetnext_event*( )
posix_trace_get_attr( )	*posix_trace_trid_eventid_open*( )
posix_trace_get_status( )	*posix_trace_trygetnext_event*( )

TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid* argument:

posix_trace_get_filter( ) *posix_trace_set_filter*( )

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

In particular, notice that the operations normally used by a trace analyzer process, such as *posix_trace_rewind*( ) or *posix_trace_close*( ), cannot be invoked using the trace stream identifier returned by the *posix_trace_create*( ) function.

TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is supported, its trace event type filter shall be empty.

The *posix_trace_create*( ) function may be called multiple times from the same or different processes, with the system-wide limit indicated by the runtime invariant value {TRACE_SYS_MAX}, which has the minimum value {_POSIX_TRACE_SYS_MAX}.

The trace stream identifier returned by the *posix_trace_create*( ) function in the argument pointed to by *trid* is valid only in the process that made the function call. If it is used from another process, that is a child process, in functions defined in IEEE Std 1003.1-2001, these functions shall return with the error [EINVAL].

TRL The *posix_trace_create_withlog*( ) function shall be equivalent to *posix_trace_create*( ), except that it associates a trace log with this stream. The *file_desc* argument shall be the file descriptor designating the trace log destination. The function shall fail if this file descriptor refers to a file with a file type that is not compatible with the log policy associated with the trace log. The list of the appropriate file types that are compatible with each log policy is implementation-defined.

The *posix_trace_create_withlog*( ) function shall return in the parameter pointed to by *trid* the trace stream identifier, which uniquely identifies the newly created trace stream, and shall be used in subsequent calls to control tracing. The *trid* argument may only be used by the following functions:

posix_trace_clear( )	*posix_trace_getnext_event*( )
posix_trace_eventid_equal( )	*posix_trace_shutdown*( )
posix_trace_eventid_get_name( )	*posix_trace_start*( )
posix_trace_eventtypelist_getnext_id( )	*posix_trace_stop*( )
posix_trace_eventtypelist_rewind( )	*posix_trace_timedgetnext_event*( )
posix_trace_flush( )	*posix_trace_trid_eventid_open*( )
posix_trace_get_attr( )	*posix_trace_trygetnext_event*( )
posix_trace_get_status( )	

TRL TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid* argument:

posix_trace_get_filter( ) *posix_trace_set_filter*( )

TRL In particular, notice that the operations normally used by a trace analyzer process, such as *posix_trace_rewind*( ) or *posix_trace_close*( ), cannot be invoked using the trace stream identifier returned by the *posix_trace_create_withlog*( ) function.

The *posix_trace_flush*( ) function shall initiate a flush operation which copies the contents of the trace stream identified by the argument *trid* into the trace log associated with the trace stream at the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this function shall return an error. The termination of the flush operation can be polled by the *posix_trace_get_status*( ) function. During the flush operation, it shall be possible to trace new trace events up to the point when the trace stream becomes full. After flushing is completed, the space used by the flushed trace events shall be available for tracing new trace events.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If flushing the trace stream causes the resulting trace log to become full, the trace log full policy shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:

POSIX_TRACE_UNTIL_FULL
: The trace events that have not yet been flushed shall be discarded.

POSIX_TRACE_LOOP
: The trace events that have not yet been flushed shall be written to the beginning of the trace log, overwriting previous trace events stored there.

POSIX_TRACE_APPEND
: The trace events that have not yet been flushed shall be appended to the trace log.

The *posix_trace_shutdown*( ) function shall stop the tracing of trace events in the trace stream identified by *trid*, as if *posix_trace_stop*( ) had been invoked. The *posix_trace_shutdown*( ) function shall free all the resources associated with the trace stream.

The *posix_trace_shutdown*( ) function shall not return until all the resources associated with the trace stream have been freed. When the *posix_trace_shutdown*( ) function returns, the *trid* argument becomes an invalid trace stream identifier. A call to this function shall unconditionally deallocate the resources regardless of whether all trace events have been retrieved by the analyzer process. Any thread blocked on one of the *trace_getnext_event*( ) functions (which specified this *trid*) before this call is unblocked with the error [EINVAL].

If the process exits, invokes a member of the *exec* family of functions, or is terminated, the trace streams that the process had created and that have not yet been shut down, shall be automatically shut down as if an explicit call were made to the *posix_trace_shutdown*( ) function.

TRL For an active trace stream with log, when the *posix_trace_shutdown*( ) function is called, all trace events that have not yet been flushed to the trace log shall be flushed, as in the *posix_trace_flush*( ) function, and the trace log shall be closed.

When a trace log is closed, all the information that may be retrieved later from the trace log through the trace interface shall have been written to the trace log. This information includes the trace attributes, the list of trace event types (with the mapping between trace event names and trace event type identifiers), and the trace status.

In addition, unspecified information shall be written to the trace log to allow detection of a valid trace log during the *posix_trace_open*( ) operation.

The *posix_trace_shutdown*( ) function shall not return until all trace events have been flushed.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

TRL The *posix_trace_create*( ) and *posix_trace_create_withlog*( ) functions store the trace stream identifier value in the object pointed to by *trid*, if successful.

ERRORS

TRL The *posix_trace_create*( ) and *posix_trace_create_withlog*( ) functions shall fail if:

[EAGAIN] No more trace streams can be started now. {TRACE_SYS_MAX} has been exceeded.

[EINTR] The operation was interrupted by a signal. No trace stream was created.

[EINVAL] One or more of the trace parameters specified by the *attr* parameter is invalid.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[ENOMEM] The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.

[EPERM] The caller does not have appropriate privilege to trace the process specified by *pid*.

[ESRCH] The *pid* argument does not refer to an existing process.

TRL The *posix_trace_create_withlog*( ) function shall fail if:

[EBADF] The *file_desc* argument is not a valid file descriptor open for writing.

[EINVAL] The *file_desc* argument refers to a file with a file type that does not support the log policy associated with the trace log.

[ENOSPC] No space left on device. The device corresponding to the argument *file_desc* does not contain the space required to create this trace log.

TRL The *posix_trace_flush*( ) and *posix_trace_shutdown*( ) functions shall fail if:

[EINVAL] The value of the *trid* argument does not correspond to an active trace stream with log.

[EFBIG] The trace log file has attempted to exceed an implementation-defined maximum file size.

[ENOSPC] No space left on device.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *exec*, *posix_trace_attr_init*( ), *posix_trace_clear*( ), *posix_trace_close*( ), *posix_trace_eventid_equal*( ), *posix_trace_eventtypelist_getnext_id*( ), *posix_trace_flush*( ), *posix_trace_get_attr*( ), *posix_trace_get_filter*( ), *posix_trace_get_status*( ), *posix_trace_getnext_event*( ), *posix_trace_open*( ), *posix_trace_set_filter*( ), *posix_trace_shutdown*( ), *posix_trace_start*( ), *posix_trace_timedgetnext_event*( ), *posix_trace_trid_eventid_open*( ), *posix_trace_start*( ), *time*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<trace.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_trace_event, posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)

SYNOPSIS

TRC
```
#include <sys/types.h>
#include <trace.h>

void posix_trace_event(trace_event_id_t event_id,
    const void *restrictdata_ptr, size_t data_len);
int posix_trace_eventid_open(const char *restrict event_name,
    trace_event_id_t *restrict event_id);
```

DESCRIPTION

The *posix_trace_event*( ) function shall record the *event_id* and the user data pointed to by *data_ptr* in the trace stream into which the calling process is being traced and in which *event_id* is not filtered out. If the total size of the user trace event data represented by *data_len* is not greater than the declared maximum size for user trace event data, then the *truncation-status* attribute of the trace event recorded is POSIX_TRACE_NOT_TRUNCATED. Otherwise, the user trace event data is truncated to this declared maximum size and the *truncation-status* attribute of the trace event recorded is POSIX_TRACE_TRUNCATED_RECORD.

If there is no trace stream created for the process or if the created trace stream is not running, or if the trace event specified by *event_id* is filtered out in the trace stream, the *posix_trace_event*( ) function shall have no effect.

The *posix_trace_eventid_open*( ) function shall associate a user trace event name with a trace event type identifier for the calling process. The trace event name is the string pointed to by the argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user trace event type identifiers that can be defined for any given process is limited by the maximum value {TRACE_USER_EVENT_MAX}, which has the minimum value {POSIX_TRACE_USER_EVENT_MAX}.

If the Trace Inherit option is not supported, the *posix_trace_eventid_open*( ) function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for the traced process, and is returned in the variable pointed to by the *event_id* argument. If the user trace event name has already been mapped for the traced process, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7 (on page 79)) user trace event shall be returned.

TRI If the Trace Inherit option is supported, the *posix_trace_eventid_open*( ) function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for all the processes being traced in this same trace stream, and is returned in the variable pointed to by the *event_id* argument. If the user trace event name has already been mapped for the traced processes, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (Table 2-7 (on page 79)) user trace event shall be returned.

Note: The above procedure, together with the fact that multiple processes can only be traced into the same trace stream by inheritance, ensure that all the processes that are traced into a trace stream have the same mapping of trace event names to trace event type identifiers.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30149

30150 If there is no trace stream created, the *posix_trace_eventid_open*( ) function shall store this
30151 information for future trace streams created for this process.

30152 **RETURN VALUE**

30153 No return value is defined for the *posix_trace_event*( ) function.

30154 Upon successful completion, the *posix_trace_eventid_open*( ) function shall return a value of zero.
30155 Otherwise, it shall return the corresponding error number. The *posix_trace_eventid_open*( )
30156 function stores the trace event type identifier value in the object pointed to by *event_id*, if
30157 successful.

30158 **ERRORS**

30159 The *posix_trace_eventid_open*( ) function shall fail if:

30160 [ENAMETOOLONG]
30161 The size of the name pointed to by the *event_name* argument was longer than
30162 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

30163 **EXAMPLES**

30164 None.

30165 **APPLICATION USAGE**

30166 None.

30167 **RATIONALE**

30168 None.

30169 **FUTURE DIRECTIONS**

30170 None.

30171 **SEE ALSO**

30172 Table 2-7 (on page 79), *posix_trace_start*( ), *posix_trace_trid_eventid_open*( ), the Base Definitions
30173 volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<trace.h>**

30174 **CHANGE HISTORY**

30175 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30176 IEEE PASC Interpretation 1003.1 #123 is applied.

30177 IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of
30178 the *posix_trace_eventid_open*( ) function and the *event_id* argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

posix_trace_eventid_equal, posix_trace_eventid_get_name, posix_trace_trid_eventid_open — manipulate the trace event type identifier (**TRACING**)

SYNOPSIS

TRC
```
#include <trace.h>

int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
    trace_event_id_t event2);
int posix_trace_eventid_get_name(trace_id_t trid,
    trace_event_id_t event, char *event_name);
TRC TEF int posix_trace_trid_eventid_open(trace_id_t trid,
    const char *restrict event_name,
    trace_event_id_t *restrict event);
```

DESCRIPTION

The *posix_trace_eventid_equal*( ) function shall compare the trace event type identifiers *event1* and *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the trace event type identifiers *event1* and *event2* are from different trace streams, the return value shall be unspecified.

The *posix_trace_eventid_get_name*( ) function shall return, in the argument pointed to by *event_name*, the trace event name associated with the trace event type identifier identified by the argument *event*, for the trace stream or for the trace log identified by the *trid* argument. The name of the trace event shall have a maximum of {TRACE_EVENT_NAME_MAX} characters (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). Successive calls to this function with the same trace event type identifier and the same trace stream identifier shall return the same event name.

TEF The *posix_trace_trid_eventid_open*( ) function shall associate a user trace event name with a trace event type identifier for a given trace stream. The trace stream is identified by the *trid* argument, and it shall be an active trace stream. The trace event name is the string pointed to by the argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user trace event type identifiers that can be defined for any given process is limited by the maximum value {TRACE_USER_EVENT_MAX}, which has the minimum value {_POSIX_TRACE_USER_EVENT_MAX}.

If the Trace Inherit option is not supported, the *posix_trace_trid_eventid_open*( ) function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for the process being traced in the trace stream identified by the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user trace event name has already been mapped for the traced process, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7 (on page 79)) user trace event shall be returned.

TEF TRI If the Trace Inherit option is supported, the *posix_trace_trid_eventid_open*( ) function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for all the processes being traced in the trace stream identified by the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user trace event name has already been mapped for the traced processes, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30228 POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7 (on page 79)) user trace event shall be
30229 returned.

30230 **RETURN VALUE**

30231 TEF Upon successful completion, the *posix_trace_eventid_get_name*( ) and
30232 *posix_trace_trid_eventid_open*( ) functions shall return a value of zero. Otherwise, they shall return
30233 the corresponding error number.

30234 The *posix_trace_eventid_equal*( ) function shall return a non-zero value if *event1* and *event2* are
30235 equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or
30236 *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*
30237 is invalid, the behavior shall be unspecified.

30238 The *posix_trace_eventid_get_name*( ) function stores the trace event name value in the object
30239 pointed to by *event_name*, if successful.

30240 TEF The *posix_trace_trid_eventid_open*( ) function stores the trace event type identifier value in the
30241 object pointed to by *event*, if successful.

30242 **ERRORS**

30243 TEF The *posix_trace_eventid_get_name*( ) and *posix_trace_trid_eventid_open*( ) functions shall fail if:

30244 [EINVAL] The *trid* argument was not a valid trace stream identifier.

30245 TEF The *posix_trace_trid_eventid_open*( ) function shall fail if:

30246 TEF [ENAMETOOLONG]
30247 The size of the name pointed to by the *event_name* argument was longer than
30248 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

30249 The *posix_trace_eventid_get_name*( ) function shall fail if:

30250 [EINVAL] The trace event type identifier *event* was not associated with any name.

30251 **EXAMPLES**

30252 None.

30253 **APPLICATION USAGE**

30254 None.

30255 **RATIONALE**

30256 None.

30257 **FUTURE DIRECTIONS**

30258 None.

30259 **SEE ALSO**

30260 Table 2-7 (on page 79), *posix_trace_event*( ), *posix_trace_getnext_event*( ), the Base Definitions
30261 volume of IEEE Std 1003.1-2001, **<trace.h>**

30262 **CHANGE HISTORY**

30263 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30264 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30265 **NAME**

30266 posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)

30267 **SYNOPSIS**

30268 TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_eventid_open(const char *restrict event_name,
    trace_event_id_t *restrict event_id);
```
30272

30273 **DESCRIPTION**

30274 Refer to *posix_trace_event*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30275 **NAME**

30276 posix_trace_eventset_add, posix_trace_eventset_del, posix_trace_eventset_empty,
30277 posix_trace_eventset_fill, posix_trace_eventset_ismember — manipulate trace event type sets
30278 (**TRACING**)

30279 **SYNOPSIS**

30280 TRC TEF
```
#include <trace.h>

int posix_trace_eventset_add(trace_event_id_t event_id,
    trace_event_set_t *set);
int posix_trace_eventset_del(trace_event_id_t event_id,
    trace_event_set_t *set);
int posix_trace_eventset_empty(trace_event_set_t *set);
int posix_trace_eventset_fill(trace_event_set_t *set, int what);
int posix_trace_eventset_ismember(trace_event_id_t event_id,
    const trace_event_set_t *restrict set, int *restrict ismember);
```
30289

30290 **DESCRIPTION**

30291 These primitives manipulate sets of trace event types. They operate on data objects addressable
30292 by the application, not on the current trace event filter of any trace stream.

30293 The *posix_trace_eventset_add*( ) and *posix_trace_eventset_del*( ) functions, respectively, shall add or
30294 delete the individual trace event type specified by the value of the argument *event_id* to or from
30295 the trace event type set pointed to by the argument *set*. Adding a trace event type already in the
30296 set or deleting a trace event type not in the set shall not be considered an error.

30297 The *posix_trace_eventset_empty*( ) function shall initialize the trace event type set pointed to by
30298 the *set* argument such that all trace event types defined, both system and user, shall be excluded
30299 from the set.

30300 The *posix_trace_eventset_fill*( ) function shall initialize the trace event type set pointed to by the
30301 argument *set*, such that the set of trace event types defined by the argument *what* shall be
30302 included in the set. The value of the argument *what* shall consist of one of the following values,
30303 as defined in the **<trace.h>** header:

30304 POSIX_TRACE_WOPID_EVENTS
30305 All the process-independent implementation-defined system trace event types are included
30306 in the set.

30307 POSIX_TRACE_SYSTEM_EVENTS
30308 All the implementation-defined system trace event types are included in the set, as are those
30309 defined in IEEE Std 1003.1-2001.

30310 POSIX_TRACE_ALL_EVENTS
30311 All trace event types defined, both system and user, are included in the set.

30312 Applications shall call either *posix_trace_eventset_empty*( ) or *posix_trace_eventset_fill*( ) at least
30313 once for each object of type **trace_event_set_t** prior to any other use of that object. If such an
30314 object is not initialized in this way, but is nonetheless supplied as an argument to any of the
30315 *posix_trace_eventset_add*( ), *posix_trace_eventset_del*( ), or *posix_trace_eventset_ismember*( ) functions,
30316 the results are undefined.

30317 The *posix_trace_eventset_ismember*( ) function shall test whether the trace event type specified by
30318 the value of the argument *event_id* is a member of the set pointed to by the argument *set*. The
30319 value returned in the object pointed to by *ismember* argument is zero if the trace event type
30320 identifier is not a member of the set and a value different from zero if it is a member of the set.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30321 **RETURN VALUE**

30322 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30323 return the corresponding error number.

30324 **ERRORS**

30325 These functions may fail if:

30326 [EINVAL] The value of one of the arguments is invalid.

30327 **EXAMPLES**

30328 None.

30329 **APPLICATION USAGE**

30330 None.

30331 **RATIONALE**

30332 None.

30333 **FUTURE DIRECTIONS**

30334 None.

30335 **SEE ALSO**

30336 *posix_trace_set_filter*( ), *posix_trace_trid_eventid_open*( ), the Base Definitions volume of
30337 IEEE Std 1003.1-2001, **<trace.h>**

30338 **CHANGE HISTORY**

30339 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind — iterate over a mapping of trace event types (**TRACING**)

SYNOPSIS

TRC
```
#include <trace.h>

int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
    trace_event_id_t *restrict event, int *restrict unavailable);
int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

DESCRIPTION

The first time *posix_trace_eventtypelist_getnext_id*( ) is called, the function shall return in the variable pointed to by *event* the first trace event type identifier of the list of trace events of the trace stream identified by the *trid* argument. Successive calls to *posix_trace_eventtypelist_getnext_id*( ) return in the variable pointed to by *event* the next trace event type identifier in that same list. Each time a trace event type identifier is successfully written into the variable pointed to by the *event* argument, the variable pointed to by the *unavailable* argument shall be set to zero. When no more trace event type identifiers are available, and so none is returned, the variable pointed to by the *unavailable* argument shall be set to a value different from zero.

The *posix_trace_eventtypelist_rewind*( ) function shall reset the next trace event type identifier to be read to the first trace event type identifier from the list of trace events used in the trace stream identified by *trid*.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_eventtypelist_getnext_id*( ) function stores the trace event type identifier value in the object pointed to by *event*, if successful.

ERRORS

These functions shall fail if:

[EINVAL] The *trid* argument was not a valid trace stream identifier.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_trace_event( ), *posix_trace_getnext_event*( ), *posix_trace_trid_eventid_open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<trace.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30384 **NAME**

30385 posix_trace_flush — trace stream flush from a process (**TRACING**)

30386 **SYNOPSIS**

30387 TRC
```
#include <sys/types.h>
```
30388
```
#include <trace.h>
```

30389 TRC TRL
```
int posix_trace_flush(trace_id_t trid);
```

30390

30391 **DESCRIPTION**

30392 Refer to *posix_trace_create*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_trace_get_attr, posix_trace_get_status — retrieve the trace attributes or trace status (**TRACING**)

SYNOPSIS

TRC
```
#include <trace.h>

int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
int posix_trace_get_status(trace_id_t trid,
    struct posix_trace_status_info *statusinfo);
```

DESCRIPTION

The *posix_trace_get_attr*( ) function shall copy the attributes of the active trace stream identified by *trid* into the object pointed to by the *attr* argument. TRL If the Trace Log option is supported, *trid* may represent a pre-recorded trace log.

The *posix_trace_get_status*( ) function shall return, in the structure pointed to by the *statusinfo* argument, the current trace status for the trace stream identified by the *trid* argument. These status values returned in the structure pointed to by *statusinfo* shall have been appropriately read to ensure that the returned values are consistent. TRL If the Trace Log option is supported and the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the completed trace stream.

Each time the *posix_trace_get_status*( ) function is used, the overrun status of the trace stream shall be reset to POSIX_TRACE_NO_OVERRUN immediately after the call completes. TRL If the Trace Log option is supported, the *posix_trace_get_status*( ) function shall behave the same as when the option is not supported except for the following differences:

- If the *trid* argument refers to a trace stream with log, each time the *posix_trace_get_status*( ) function is used, the log overrun status of the trace stream shall be reset to POSIX_TRACE_NO_OVERRUN and the *flush_error* status shall be reset to zero immediately after the call completes.

- If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the status of the completed trace stream and the status values of the trace stream shall not be reset.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_get_attr*( ) function stores the trace attributes in the object pointed to by *attr*, if successful.

The *posix_trace_get_status*( ) function stores the trace status in the object pointed to by *statusinfo*, if successful.

ERRORS

These functions shall fail if:

[EINVAL] The trace stream argument *trid* does not correspond to a valid active trace stream or a valid trace log.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30435 **EXAMPLES**

30436 None.

30437 **APPLICATION USAGE**

30438 None.

30439 **RATIONALE**

30440 None.

30441 **FUTURE DIRECTIONS**

30442 None.

30443 **SEE ALSO**

30444 *posix_trace_attr_destroy*( ), *posix_trace_attr_init*( ), *posix_trace_create*( ), *posix_trace_open*( ), the Base
30445 Definitions volume of IEEE Std 1003.1-2001, **<trace.h>**

30446 **CHANGE HISTORY**

30447 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30448 IEEE PASC Interpretation 1003.1 #123 is applied.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30449 **NAME**

30450 posix_trace_get_filter, posix_trace_set_filter — retrieve and set the filter of an initialized trace
30451 stream (**TRACING**)

30452 **SYNOPSIS**

30453 TRC TEF
```
#include <trace.h>

int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
int posix_trace_set_filter(trace_id_t trid,
    const trace_event_set_t *set, int how);
```
30457

30458 **DESCRIPTION**

30459 The *posix_trace_get_filter*( ) function shall retrieve, into the argument pointed to by *set*, the actual
30460 trace event filter from the trace stream specified by *trid*.

30461 The *posix_trace_set_filter*( ) function shall change the set of filtered trace event types after a trace
30462 stream identified by the *trid* argument is created. This function may be called prior to starting
30463 the trace stream, or while the trace stream is active. By default, if no call is made to
30464 *posix_trace_set_filter*( ), all trace events shall be recorded (that is, none of the trace event types are
30465 filtered out).

30466 If this function is called while the trace is in progress, a special system trace event,
30467 POSIX_TRACE_FILTER, shall be recorded in the trace indicating both the old and the new sets
30468 of filtered trace event types (see Table 2-4 (on page 78) and Table 2-6 (on page 79)).

30469 If the *posix_trace_set_filter*( ) function is interrupted by a signal, an error shall be returned and the
30470 filter shall not be changed. In this case, the state of the trace stream shall not be changed.

30471 The value of the argument *how* indicates the manner in which the set is to be changed and shall
30472 have one of the following values, as defined in the **<trace.h>** header:

30473 POSIX_TRACE_SET_EVENTSET
30474 The resulting set of trace event types to be filtered shall be the trace event type set pointed
30475 to by the argument *set*.

30476 POSIX_TRACE_ADD_EVENTSET
30477 The resulting set of trace event types to be filtered shall be the union of the current set and
30478 the trace event type set pointed to by the argument *set*.

30479 POSIX_TRACE_SUB_EVENTSET
30480 The resulting set of trace event types to be filtered shall be all trace event types in the
30481 current set that are not in the set pointed to by the argument *set*; that is, remove each
30482 element of the specified set from the current filter.

30483 **RETURN VALUE**

30484 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30485 return the corresponding error number.

30486 The *posix_trace_get_filter*( ) function stores the set of filtered trace event types in *set*, if successful.

30487 **ERRORS**

30488 These functions shall fail if:

30489 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream
30490 or the value of the argument pointed to by *set* is invalid.

30491 [EINTR] The operation was interrupted by a signal.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30492 **EXAMPLES**

30493 None.

30494 **APPLICATION USAGE**

30495 None.

30496 **RATIONALE**

30497 None.

30498 **FUTURE DIRECTIONS**

30499 None.

30500 **SEE ALSO**

30501 Table 2-4 (on page 78), Table 2-6 (on page 79), *posix_trace_eventset_add*( ), the Base Definitions
30502 volume of IEEE Std 1003.1-2001, **<trace.h>**

30503 **CHANGE HISTORY**

30504 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30505 IEEE PASC Interpretation 1003.1 #123 is applied.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30506 **NAME**

30507 posix_trace_get_status — retrieve the trace status (**TRACING**)

30508 **SYNOPSIS**

30509 TRC
```
#include <trace.h>

int posix_trace_get_status(trace_id_t trid,
    struct posix_trace_status_info *statusinfo);
```
30510

30511

30512

30513 **DESCRIPTION**

30514 Refer to *posix_trace_get_attr*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

posix_trace_getnext_event, posix_trace_timedgetnext_event, posix_trace_trygetnext_event — retrieve a trace event (**TRACING**)

SYNOPSIS

TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_getnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```
TRC TMO
```
int posix_trace_timedgetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable,
    const struct timespec *restrict abs_timeout);
```
TRC
```
int posix_trace_trygetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```

DESCRIPTION

The *posix_trace_getnext_event*( ) function shall report a recorded trace event either from an active trace stream without log TRL or a pre-recorded trace stream identified by the *trid* argument. The *posix_trace_trygetnext_event*( ) function shall report a recorded trace event from an active trace stream without log identified by the *trid* argument.

The trace event information associated with the recorded trace event shall be copied by the function into the structure pointed to by the argument *event* and the data associated with the trace event shall be copied into the buffer pointed to by the *data* argument.

The *posix_trace_getnext_event*( ) function shall block if the *trid* argument identifies an active trace stream and there is currently no trace event ready to be retrieved. When returning, if a recorded trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different from zero.

TMO The *posix_trace_timedgetnext_event*( ) function shall attempt to get another trace event from an active trace stream without log, as in the *posix_trace_getnext_event*( ) function. However, if no trace event is available from the trace stream, the implied wait shall be terminated when the timeout specified by the argument *abs_timeout* expires, and the function shall return the error [ETIMEDOUT].

The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by the clock upon which timeouts are based (that is, when the value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout* has already passed at the time of the call.

TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock; if the Timers option is not supported, the timeout shall be based on the system clock as returned by the *time*( ) function. The resolution of the timeout shall be the resolution of the clock on which it is based. The **timespec** data type is defined in the **<time.h>** header.

TMO Under no circumstance shall the function fail with a timeout if a trace event is immediately available from the trace stream. The validity of the *abs_timeout* argument need not be checked if

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

a trace event is immediately available from the trace stream.

The behavior of this function for a pre-recorded trace stream is unspecified.

TRL The *posix_trace_trygetnext_event*( ) function shall not block. This function shall return an error if the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace event was reported, the variable pointed to by the *unavailable* argument shall be set to a value different from zero.

The argument *num_bytes* shall be the size of the buffer pointed to by the *data* argument. The argument *data_len* reports to the application the length in bytes of the data record just transferred. If *num_bytes* is greater than or equal to the size of the data associated with the trace event pointed to by the *event* argument, all the recorded data shall be transferred. In this case, the *truncation-status* member of the trace event structure shall be either POSIX_TRACE_NOT_TRUNCATED, if the trace event data was recorded without truncation while tracing, or POSIX_TRACE_TRUNCATED_RECORD, if the trace event data was truncated when it was recorded. If the *num_bytes* argument is less than the length of recorded trace event data, the data transferred shall be truncated to a length of *num_bytes*, the value stored in the variable pointed to by *data_len* shall be equal to *num_bytes*, and the *truncation-status* member of the *event* structure argument shall be set to POSIX_TRACE_TRUNCATED_READ (see the **posix_trace_event_info** structure defined in **<trace.h>**).

The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace events shall be reported in the order in which they were generated, up to an implementation-defined time resolution that causes the ordering of trace events occurring very close to each other to be unknown. Once reported, a trace event cannot be reported again from an active trace stream. Once a trace event is reported from an active trace stream without log, the trace stream shall make the resources associated with that trace event available to record future generated trace events.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, these functions store:

- The recorded trace event in the object pointed to by *event*
- The trace event information associated with the recorded trace event in the object pointed to by *data*
- The length of this trace event information in the object pointed to by *data_len*
- The value of zero in the object pointed to by *unavailable*

ERRORS

These functions shall fail if:

[EINVAL] The trace stream identifier argument *trid* is invalid.

The *posix_trace_getnext_event*( ) and *posix_trace_timedgetnext_event*( ) functions shall fail if:

[EINTR] The operation was interrupted by a signal, and so the call had no effect.

The *posix_trace_trygetnext_event*( ) function shall fail if:

[EINVAL] The trace stream identifier argument *trid* does not correspond to an active trace stream.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30606 TMO The *posix_trace_timedgetnext_event*( ) function shall fail if:

30607 [EINVAL] There is no trace event immediately available from the trace stream, and the
30608 *timeout* argument is invalid.

30609 [ETIMEDOUT] No trace event was available from the trace stream before the specified
30610 timeout *timeout* expired.
30611

30612 **EXAMPLES**

30613 None.

30614 **APPLICATION USAGE**

30615 None.

30616 **RATIONALE**

30617 None.

30618 **FUTURE DIRECTIONS**

30619 None.

30620 **SEE ALSO**

30621 *posix_trace_create*( ), *posix_trace_open*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
30622 **<sys/types.h>**, **<trace.h>**

30623 **CHANGE HISTORY**

30624 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30625 IEEE PASC Interpretation 1003.1 #123 is applied.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30626 **NAME**

30627 posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)

30628 **SYNOPSIS**

30629 TRC TRL
```
#include <trace.h>

int posix_trace_open(int file_desc, trace_id_t *trid);
int posix_trace_rewind(trace_id_t trid);
```
30630

30631

30632

30633 **DESCRIPTION**

30634 Refer to *posix_trace_close*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

30635 **NAME**

30636 posix_trace_set_filter — set filter of an initialized trace stream (**TRACING**)

30637 **SYNOPSIS**

30638 TRC TEF
```
#include <trace.h>

int posix_trace_set_filter(trace_id_t trid,
    const trace_event_set_t *set, int how);
```
30641

30642 **DESCRIPTION**

30643 Refer to *posix_trace_get_filter*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30644 **NAME**

30645 posix_trace_shutdown — trace stream shutdown from a process (**TRACING**)

30646 **SYNOPSIS**

30647 TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_shutdown(trace_id_t trid);
```
30650

30651 **DESCRIPTION**

30652 Refer to *posix_trace_create*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30653 **NAME**

30654 posix_trace_start, posix_trace_stop — trace start and stop (**TRACING**)

30655 **SYNOPSIS**

30656 TRC
```
#include <trace.h>

int posix_trace_start(trace_id_t trid);
int posix_trace_stop (trace_id_t trid);
```
30659

30660 **DESCRIPTION**

30661 The *posix_trace_start*( ) and *posix_trace_stop*( ) functions, respectively, shall start and stop the
30662 trace stream identified by the argument *trid*.

30663 The effect of calling the *posix_trace_start*( ) function shall be recorded in the trace stream as the
30664 POSIX_TRACE_START system trace event and the status of the trace stream shall become
30665 POSIX_TRACE_RUNNING. If the trace stream is in progress when this function is called, the
30666 POSIX_TRACE_START system trace event shall not be recorded and the trace stream shall
30667 continue to run. If the trace stream is full, the POSIX_TRACE_START system trace event shall
30668 not be recorded and the status of the trace stream shall not be changed.

30669 The effect of calling the *posix_trace_stop*( ) function shall be recorded in the trace stream as the
30670 POSIX_TRACE_STOP system trace event and the status of the trace stream shall become
30671 POSIX_TRACE_SUSPENDED. If the trace stream is suspended when this function is called, the
30672 POSIX_TRACE_STOP system trace event shall not be recorded and the trace stream shall remain
30673 suspended. If the trace stream is full, the POSIX_TRACE_STOP system trace event shall not be
30674 recorded and the status of the trace stream shall not be changed.

30675 **RETURN VALUE**

30676 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30677 return the corresponding error number.

30678 **ERRORS**

30679 These functions shall fail if:

30680 [EINVAL] The value of the argument *trid* does not correspond to an active trace stream
30681 and thus no trace stream was started or stopped.

30682 [EINTR] The operation was interrupted by a signal and thus the trace stream was not
30683 necessarily started or stopped.

30684 **EXAMPLES**

30685 None.

30686 **APPLICATION USAGE**

30687 None.

30688 **RATIONALE**

30689 None.

30690 **FUTURE DIRECTIONS**

30691 None.

30692 **SEE ALSO**

30693 *posix_trace_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<trace.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30694 **CHANGE HISTORY**

30695 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30696 IEEE PASC Interpretation 1003.1 #123 is applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30697 **NAME**

30698 posix_trace_timedgetnext_event, — retrieve a trace event (**TRACING**)

30699 **SYNOPSIS**

30700 TRC TMO
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_timedgetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable,
    const struct timespec *restrict abs_timeout);
```
30707

30708 **DESCRIPTION**

30709 Refer to *posix_trace_getnext_event*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_trace_trid_eventid_open — open a trace event type identifier (**TRACING**)

SYNOPSIS

TRC TEF
```
#include <trace.h>

int posix_trace_trid_eventid_open(trace_id_t trid,
    const char *restrict event_name,
    trace_event_id_t *restrict event);
```

DESCRIPTION

Refer to *posix_trace_eventid_equal*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30720 **NAME**

30721 posix_trace_trygetnext_event — retrieve a trace event (**TRACING**)

30722 **SYNOPSIS**

30723 TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_trygetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```
30729

30730 **DESCRIPTION**

30731 Refer to *posix_trace_getnext_event*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30732 **NAME**

30733 posix_typed_mem_get_info — query typed memory information (**ADVANCED REALTIME**)

30734 **SYNOPSIS**

30735 TYM
```
#include <sys/mman.h>
```

30736
```
int posix_typed_mem_get_info(int fildes,
    struct posix_typed_mem_info *info);
```

30738

30739 **DESCRIPTION**

30740 The *posix_typed_mem_get_info*( ) function shall return, in the *posix_tmi_length* field of the
30741 **posix_typed_mem_info** structure pointed to by *info*, the maximum length which may be
30742 successfully allocated by the typed memory object designated by *fildes*. This maximum length
30743 shall take into account the flag POSIX_TYPED_MEM_ALLOCATE or
30744 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified when the typed memory object
30745 represented by *fildes* was opened. The maximum length is dynamic; therefore, the value returned
30746 is valid only while the current mapping of the corresponding typed memory pool remains
30747 unchanged.

30748 If *fildes* represents a typed memory object opened with neither the
30749 POSIX_TYPED_MEM_ALLOCATE flag nor the POSIX_TYPED_MEM_ALLOCATE_CONTIG
30750 flag specified, the returned value of *info->posix_tmi_length* is unspecified.

30751 The *posix_typed_mem_get_info*( ) function may return additional implementation-defined
30752 information in other fields of the **posix_typed_mem_info** structure pointed to by *info*.

30753 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this
30754 function is undefined.

30755 **RETURN VALUE**

30756 Upon successful completion, the *posix_typed_mem_get_info*( ) function shall return zero;
30757 otherwise, the corresponding error status value shall be returned.

30758 **ERRORS**

30759 The *posix_typed_mem_get_info*( ) function shall fail if:

30760 [EBADF] The *fildes* argument is not a valid open file descriptor.

30761 [ENODEV] The *fildes* argument is not connected to a memory object supported by this
30762 function.

30763 This function shall not return an error code of [EINTR].

30764 **EXAMPLES**

30765 None.

30766 **APPLICATION USAGE**

30767 None.

30768 **RATIONALE**

30769 An application that needs to allocate a block of typed memory with length dependent upon the
30770 amount of memory currently available must either query the typed memory object to obtain the
30771 amount available, or repeatedly invoke *mmap*( ) attempting to guess an appropriate length.
30772 While the latter method is existing practice with *malloc*( ), it is awkward and imprecise. The
30773 *posix_typed_mem_get_info*( ) function allows an application to immediately determine available
30774 memory. This is particularly important for typed memory objects that may in some cases be
30775 scarce resources. Note that when a typed memory pool is a shared resource, some form of
30776 mutual-exclusion or synchronization may be required while typed memory is being queried and

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30777 allocated to prevent race conditions.

30778 The existing *fstat*( ) function is not suitable for this purpose. We realize that implementations
30779 may wish to provide other attributes of typed memory objects (for example, alignment
30780 requirements, page size, and so on). The *fstat*( ) function returns a structure which is not
30781 extensible and, furthermore, contains substantial information that is inappropriate for typed
30782 memory objects.

30783 **FUTURE DIRECTIONS**

30784 None.

30785 **SEE ALSO**

30786 *fstat*( ), *mmap*( ), *posix_typed_mem_open*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
30787 **<sys/mman.h>**

30788 **CHANGE HISTORY**

30789 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

posix_typed_mem_open — open a typed memory object (**ADVANCED REALTIME**)

SYNOPSIS

TYM
```
#include <sys/mman.h>

int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

DESCRIPTION

The *posix_typed_mem_open*( ) function shall establish a connection between the typed memory object specified by the string pointed to by *name* and a file descriptor. It shall create an open file description that refers to the typed memory object and a file descriptor that refers to that open file description. The file descriptor is used by other functions to refer to that typed memory object. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument shall conform to the construction rules for a pathname. If *name* begins with the slash character, then processes calling *posix_typed_mem_open*( ) with the same value of *name* shall refer to the same typed memory object. If *name* does not begin with the slash character, the effect is implementation-defined. The interpretation of slash characters other than the leading slash character in *name* is implementation-defined.

Each typed memory object supported in a system shall be identified by a name which specifies not only its associated typed memory pool, but also the path or port by which it is accessed. That is, the same typed memory pool accessed via several different ports shall have several different corresponding names. The binding between names and typed memory objects is established in an implementation-defined manner. Unlike shared memory objects, there is no way within IEEE Std 1003.1-2001 for a program to create a typed memory object.

The value of *tflag* shall determine how the typed memory object behaves when subsequently mapped by calls to *mmap*( ). At most, one of the following flags defined in **<sys/mman.h>** may be specified:

POSIX_TYPED_MEM_ALLOCATE
: Allocate on *mmap*( ).

POSIX_TYPED_MEM_ALLOCATE_CONTIG
: Allocate contiguously on *mmap*( ).

POSIX_TYPED_MEM_MAP_ALLOCATABLE
: Map on *mmap*( ), without affecting allocatability.

If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE specified, any subsequent call to *mmap*( ) using the returned file descriptor shall result in allocation and mapping of typed memory from the specified typed memory pool. The allocated memory may be a contiguous previously unallocated area of the typed memory pool or several non-contiguous previously unallocated areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap*( ) using the returned file descriptor shall result in allocation and mapping of a single contiguous previously unallocated area of the typed memory pool (also mapped to a contiguous portion of the process address space). If *tflag* has none of the flags POSIX_TYPED_MEM_ALLOCATE or POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap*( ) using the returned file descriptor shall map an application-chosen area from the specified typed memory pool such that this mapped area becomes unavailable for allocation until unmapped by all processes. If *tflag* has the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE specified, any subsequent call to *mmap*( ) using the returned file descriptor shall map an application-chosen area from the specified typed memory pool without an effect on the availability of that area for

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. The appropriate privilege to specify the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag is implementation-defined.

If successful, *posix_typed_mem_open*( ) shall return a file descriptor for the typed memory object that is the lowest numbered file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor shall not share it with any other processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared.

The behavior of *msync*( ), *ftruncate*( ), and all file operations other than *mmap*( ), *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), *fstat*( ), *dup*( ), *dup2*( ), and *close*( ), is unspecified when passed a file descriptor connected to a typed memory object by this function.

The file status flags of the open file description shall be set according to the value of *oflag*. Applications shall specify exactly one of the three access mode values described below and defined in the **<fcntl.h>** header, as the value of *oflag*.

O_RDONLY Open for read access only.

O_WRONLY Open for write access only.

O_RDWR Open for read or write access.

RETURN VALUE

Upon successful completion, the *posix_typed_mem_open*( ) function shall return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *posix_typed_mem_open*( ) function shall fail if:

[EACCES] The typed memory object exists and the permissions specified by *oflag* are denied.

[EINTR] The *posix_typed_mem_open*( ) operation was interrupted by a signal.

[EINVAL] The flags specified in *tflag* are invalid (more than one of POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, or POSIX_TYPED_MEM_MAP_ALLOCATABLE is specified).

[EMFILE] Too many file descriptors are currently in use by this process.

[ENAMETOOLONG] The length of the *name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENFILE] Too many file descriptors are currently open in the system.

[ENOENT] The named typed memory object does not exist.

[EPERM] The caller lacks the appropriate privilege to specify the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE in argument *tflag*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30877 **EXAMPLES**

30878 None.

30879 **APPLICATION USAGE**

30880 None.

30881 **RATIONALE**

30882 None.

30883 **FUTURE DIRECTIONS**

30884 None.

30885 **SEE ALSO**

30886 *close*( ), *dup*( ), *exec*, *fcntl*( ), *fstat*( ), *ftruncate*( ), *mmap*( ), *msync*( ), *posix_mem_offset*( ),
30887 *posix_typed_mem_get_info*( ), *umask*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
30888 **<fcntl.h>**, **<sys/mman.h>**

30889 **CHANGE HISTORY**

30890 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

pow, powf, powl — power function

SYNOPSIS

```
#include <math.h>

double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the value of x raised to the power y, x^y. If x is negative, the application shall ensure that y is an integer value.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of x raised to the power y.

MX For finite values of $x < 0$, and finite non-integer values of y, a domain error shall occur and either a NaN (if representable), or an implementation-defined value shall be returned.

If the correct value would cause overflow, a range error shall occur and *pow*( ), *powf*( ), and *powl*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

If the correct value would cause underflow, and is not representable, a range error may occur, and MX either 0.0 (if supported), or an implementation-defined value shall be returned.

MX If x or y is a NaN, a NaN shall be returned (unless specified elsewhere in this description).

For any value of y (including NaN), if x is +1, 1.0 shall be returned.

For any value of x (including NaN), if y is ±0, 1.0 shall be returned.

For any odd integer value of $y > 0$, if x is ±0, ±0 shall be returned.

For $y > 0$ and not an odd integer, if x is ±0, +0 shall be returned.

If x is –1, and y is ±Inf, 1.0 shall be returned.

For $|x| < 1$, if y is –Inf, +Inf shall be returned.

For $|x| > 1$, if y is –Inf, +0 shall be returned.

For $|x| < 1$, if y is +Inf, +0 shall be returned.

For $|x| > 1$, if y is +Inf, +Inf shall be returned.

For y an odd integer < 0, if x is –Inf, –0 shall be returned.

For $y < 0$ and not an odd integer, if x is –Inf, +0 shall be returned.

For y an odd integer > 0, if x is –Inf, –Inf shall be returned.

For $y > 0$ and not an odd integer, if x is –Inf, +Inf shall be returned.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For $y < 0$, if x is +Inf, +0 shall be returned.

For $y > 0$, if x is +Inf, +Inf shall be returned.

For y an odd integer < 0, if x is ±0, a pole error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL shall be returned for *pow*( ), *powf*( ), and *powl*( ), respectively.

For $y < 0$ and not an odd integer, if x is ±0, a pole error shall occur and HUGE_VAL, HUGE_VALF, and HUGE_VALL shall be returned for *pow*( ), *powf*( ), and *powl*( ), respectively.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Domain Error — The value of x is negative and y is a finite non-integer.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

MX Pole Error — The value of x is zero and y is negative.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error — The result overflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error — The result underflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30971 **SEE ALSO**

30972 *exp*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
30973 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

30974 **CHANGE HISTORY**

30975 First released in Issue 1. Derived from Issue 1 of the SVID.

30976 **Issue 5**

30977 The DESCRIPTION is updated to indicate how an application should check for an error. This
30978 text was previously published in the APPLICATION USAGE section.

30979 **Issue 6**

30980 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

30981 The *powf*( ) and *powl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

30982 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
30983 revised to align with the ISO / IEC 9899: 1999 standard.

30984 IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are
30985 marked.

30986 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 42 is applied, correcting the third
30987 paragraph in the RETURN VALUE section.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30988 **NAME**

30989 pread — read from a file

30990 **SYNOPSIS**

30991 XSI
```
#include <unistd.h>
```

30992
```
ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
```

30993

30994 **DESCRIPTION**

30995 Refer to *read*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30996 **NAME**

30997 printf — print formatted output

30998 **SYNOPSIS**

30999 `#include <stdio.h>`

31000 `int printf(const char *restrict format, ...);`

31001 **DESCRIPTION**

31002 Refer to *fprintf*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

31003 **NAME**

31004 pselect, select — synchronous I/O multiplexing

31005 **SYNOPSIS**

31006 ```
#include <sys/select.h>
```

31007 ```
int pselect(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    const struct timespec *restrict timeout,
    const sigset_t *restrict sigmask);
int select(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    struct timeval *restrict timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

31018 **DESCRIPTION**

31019 The *pselect*( ) function shall examine the file descriptor sets whose addresses are passed in the
31020 *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for
31021 reading, are ready for writing, or have an exceptional condition pending, respectively.

31022 The *select*( ) function shall be equivalent to the *pselect*( ) function, except as follows:

31023 • For the *select*( ) function, the timeout period is given in seconds and microseconds in an
31024 argument of type **struct timeval**, whereas for the *pselect*( ) function the timeout period is
31025 given in seconds and nanoseconds in an argument of type **struct timespec**.

31026 • The *select*( ) function has no *sigmask* argument; it shall behave as *pselect*( ) does when *sigmask*
31027 is a null pointer.

31028 • Upon successful completion, the *select*( ) function may modify the object pointed to by the
31029 *timeout* argument.

31030 The *pselect*( ) and *select*( ) functions shall support regular files, terminal and pseudo-terminal
31031 XSR devices, STREAMS-based files, FIFOs, pipes, and sockets. The behavior of *pselect*( ) and *select*( )
31032 on file descriptors that refer to other types of file is unspecified.

31033 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall
31034 be checked in each set; that is, the descriptors from zero through *nfds*−1 in the descriptor sets
31035 shall be examined.

31036 If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input
31037 specifies the file descriptors to be checked for being ready to read, and on output indicates
31038 which file descriptors are ready to read.

31039 If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input
31040 specifies the file descriptors to be checked for being ready to write, and on output indicates
31041 which file descriptors are ready to write.

31042 If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input
31043 specifies the file descriptors to be checked for error conditions pending, and on output indicates
31044 which file descriptors have error conditions pending.

31045 Upon successful completion, the *pselect*( ) or *select*( ) function shall modify the objects pointed to
31046 by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for
31047 reading, ready for writing, or have an error condition pending, respectively, and shall return the
31048 total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

corresponding bit shall be set on successful completion if it was set on input and the associated condition is true for that file descriptor.

If none of the selected descriptors are ready for the requested operation, the *pselect*( ) or *select*( ) function shall block until at least one of the requested operations becomes ready, until the *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the *pselect*( ) or *select*( ) function shall take before timing out. If the *timeout* parameter is not a null pointer, it specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, the function shall return. If the *timeout* parameter is a null pointer, then the call to *pselect*( ) or *select*( ) shall block indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout* parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

The use of a timeout does not affect any pending timers set up by *alarm*( ), *ualarm*( ), or *setitimer*( ).

Implementations may place limitations on the maximum timeout interval supported. All implementations shall support a maximum timeout interval of at least 31 days. If the *timeout* argument specifies a timeout interval greater than the implementation-defined maximum value, the maximum value shall be used as the actual timeout value. Implementations may also place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.

If *sigmask* is not a null pointer, then the *pselect*( ) function shall replace the signal mask of the process by the set of signals pointed to by *sigmask* before examining the descriptors, and shall restore the signal mask of the process before returning.

A descriptor shall be considered ready for reading when a call to an input function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully. (The function might return data, an end-of-file indication, or an error other than one indicating that it is blocked, and in each of these cases the descriptor shall be considered ready for reading.)

A descriptor shall be considered ready for writing when a call to an output function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully.

If a socket has a pending error, it shall be considered to have an exceptional condition pending. Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for use with a socket, it is protocol-specific except as noted below. For other file types it is implementation-defined. If the operation is meaningless for a particular file type, *pselect*( ) or *select*( ) shall indicate that the descriptor is ready for read or write operations, and shall indicate that the descriptor has no exceptional condition pending.

If a descriptor refers to a socket, the implied input function is the *recvmsg*( ) function with parameters requesting normal and ancillary data, such that the presence of either type shall cause the socket to be marked as readable. The presence of out-of-band data shall be checked if the socket option SO_OOBINLINE has been enabled, as out-of-band data is enqueued with normal data. If the socket is currently listening, then it shall be marked as readable if an incoming connection request has been received, and a call to the *accept*( ) function shall complete without blocking.

If a descriptor refers to a socket, the implied output function is the *sendmsg*( ) function supplying an amount of normal data equal to the current value of the SO_SNDLOWAT option for the socket. If a non-blocking call to the *connect*( ) function has been made for a socket, and the connection attempt has either succeeded or failed leaving a pending error, the socket shall be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

marked as writable.

A socket shall be considered to have an exceptional condition pending if a receive operation with O_NONBLOCK clear for the open file description and with the MSG_OOB flag set would return out-of-band data without blocking. (It is protocol-specific whether the MSG_OOB flag would be used to read out-of-band data.) A socket shall also be considered to have an exceptional condition pending if an out-of-band data mark is present in the receive queue. Other circumstances under which a socket may be considered to have an exceptional condition pending are protocol-specific and implementation-defined.

If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is not a null pointer, the *pselect*( ) or *select*( ) function shall block for the time specified, or until interrupted by a signal. If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is a null pointer, the *pselect*( ) or *select*( ) function shall block until interrupted by a signal.

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall not be modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall have all bits set to 0.

File descriptor masks of type **fd_set** can be initialized and tested with *FD_CLR*( ), *FD_ISSET*( ), *FD_SET*( ), and *FD_ZERO*( ). It is unspecified whether each of these is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with any of these names, the behavior is undefined.

FD_CLR(*fd*, *fdsetp*) shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not a member of this set, there shall be no effect on the set, nor will an error be returned.

FD_ISSET(*fd*, *fdsetp*) shall evaluate to non-zero if the file descriptor *fd* is a member of the set pointed to by *fdsetp*, and shall evaluate to zero otherwise.

FD_SET(*fd*, *fdsetp*) shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file descriptor *fd* is already in this set, there shall be no effect on the set, nor will an error be returned.

FD_ZERO(*fdsetp*) shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is returned if the set is not empty at the time *FD_ZERO*( ) is invoked.

The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or equal to FD_SETSIZE, or if *fd* is not a valid file descriptor, or if any of the arguments are expressions with side effects.

RETURN VALUE

Upon successful completion, the *pselect*( ) and *select*( ) functions shall return the total number of bits set in the bit masks. Otherwise, –1 shall be returned, and *errno* shall be set to indicate the error.

FD_CLR( ), *FD_SET*( ), and *FD_ZERO*( ) do not return a value. *FD_ISSET*( ) shall return a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

ERRORS

Under the following conditions, *pselect*( ) and *select*( ) shall fail and set *errno* to:

[EBADF] One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EINTR] The function was interrupted before any of the selected events occurred and before the timeout interval expired.

XSI If SA_RESTART has been set for the interrupting signal, it is implementation-defined whether the function restarts or returns with [EINTR].

[EINVAL] An invalid timeout interval was specified.

[EINVAL] The *nfds* argument is less than 0 or greater than FD_SETSIZE.

XSR [EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In previous versions of the Single UNIX Specification, the *select*( ) function was defined in the **<sys/time.h>** header. This is now changed to **<sys/select.h>**. The rationale for this change was as follows: the introduction of the *pselect*( ) function included the **<sys/select.h>** header and the **<sys/select.h>** header defines all the related definitions for the *pselect*( ) and *select*( ) functions. Backwards-compatibility to existing XSI implementations is handled by allowing **<sys/time.h>** to include **<sys/select.h>**.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *alarm*( ), *connect*( ), *fcntl*( ), *poll*( ), *read*( ), *recvmsg*( ), *sendmsg*( ), *setitimer*( ), *ualarm*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/select.h>**, **<sys/time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when *nfds* is less than 0 or greater than FD_SETSIZE. It previously stated less than 0, or greater than or equal to FD_SETSIZE.

Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.

Issue 6

The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs* in the *select*( ) DESCRIPTION to be *readfds* and *writefds*.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- These functions are now mandatory.

The *pselect*( ) function is added for alignment with IEEE Std 1003.1g-2000 and additional detail related to sockets semantics is added to the DESCRIPTION.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *select*( ) function now requires inclusion of **<sys/select.h>**.

The **restrict** keyword is added to the *select*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31188 **NAME**

31189 pthread_atfork — register fork handlers

31190 **SYNOPSIS**

31191 THR
```
#include <pthread.h>

int pthread_atfork(void (*prepare)(void), void (*parent)(void),
    void (*child)(void));
```
31192
31193
31194

31195 **DESCRIPTION**

31196 The *pthread_atfork*( ) function shall declare fork handlers to be called before and after *fork*( ), in
31197 the context of the thread that called *fork*( ). The *prepare* fork handler shall be called before *fork*( )
31198 processing commences. The *parent* fork handle shall be called after *fork*( ) processing completes
31199 in the parent process. The *child* fork handler shall be called after *fork*( ) processing completes in
31200 the child process. If no handling is desired at one or more of these three points, the
31201 corresponding fork handler address(es) may be set to NULL.

31202 The order of calls to *pthread_atfork*( ) is significant. The *parent* and *child* fork handlers shall be
31203 called in the order in which they were established by calls to *pthread_atfork*( ). The *prepare* fork
31204 handlers shall be called in the opposite order.

31205 **RETURN VALUE**

31206 Upon successful completion, *pthread_atfork*( ) shall return a value of zero; otherwise, an error
31207 number shall be returned to indicate the error.

31208 **ERRORS**

31209 The *pthread_atfork*( ) function shall fail if:

31210 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

31211 The *pthread_atfork*( ) function shall not return an error code of [EINTR].

31212 **EXAMPLES**

31213 None.

31214 **APPLICATION USAGE**

31215 None.

31216 **RATIONALE**

31217 There are at least two serious problems with the semantics of *fork*( ) in a multi-threaded
31218 program. One problem has to do with state (for example, memory) covered by mutexes.
31219 Consider the case where one thread has a mutex locked and the state covered by that mutex is
31220 inconsistent while another thread calls *fork*( ). In the child, the mutex is in the locked state
31221 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply
31222 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about
31223 how to correct or otherwise deal with the inconsistent state in the child.

31224 It is suggested that programs that use *fork*( ) call an *exec* function very soon afterwards in the
31225 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe
31226 library routines are promised to be available.

31227 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application
31228 programs may not be aware that a multi-threaded library is in use, and they feel free to call any
31229 number of library routines between the *fork*( ) and *exec* calls, just as they always have. Indeed,
31230 they may be extant single-threaded programs and cannot, therefore, be expected to obey new
31231 restrictions imposed by the threads library.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

On the other hand, the multi-threaded library needs a way to protect its internal state during *fork*( ) in case it is re-entered later in the child process. The problem arises especially in multi-threaded I/O libraries, which are almost sure to be invoked between the *fork*( ) and *exec* calls to effect I/O redirection. The solution may require locking mutex variables during *fork*( ), or it may entail simply resetting the state in the child after the *fork*( ) processing completes.

The *pthread_atfork*( ) function provides multi-threaded libraries with a means to protect themselves from innocent application programs that call *fork*( ), and it provides multi-threaded application programs with a standard mechanism for protecting themselves from *fork*( ) calls in a library routine or the application itself.

The expected usage is that the *prepare* handler acquires all mutex locks and the other two fork handlers release them.

For example, an application can supply a *prepare* routine that acquires the necessary mutexes the library maintains and supply *child* and *parent* routines that release those mutexes, thus ensuring that the child gets a consistent snapshot of the state of the library (and that no mutexes are left stranded). Alternatively, some libraries might be able to supply just a *child* routine that reinitializes the mutexes in the library and all associated states to some known value (for example, what it was when the image was originally executed).

When *fork*( ) is called, only the calling thread is duplicated in the child process. Synchronization variables remain in the same state in the child as they were in the parent at the time *fork*( ) was called. Thus, for example, mutex locks may be held by threads that no longer exist in the child process, and any associated states may be inconsistent. The parent process may avoid this by explicit code that acquires and releases locks critical to the child via *pthread_atfork*( ). In addition, any critical threads need to be recreated and reinitialized to the proper state in the child (also via *pthread_atfork*( )).

A higher-level package may acquire locks on its own data structures before invoking lower-level packages. Under this scenario, the order specified for fork handler calls allows a simple rule of initialization for avoiding package deadlock: a package initializes all packages on which it depends before it calls the *pthread_atfork*( ) function for itself.

FUTURE DIRECTIONS

None.

SEE ALSO

atexit( ), *fork*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 5. Derived from the POSIX Threads Extension.

IEEE PASC Interpretation 1003.1c #4 is applied.

Issue 6

The *pthread_atfork*( ) function is marked as part of the Threads option.

The **<pthread.h>** header is added to the SYNOPSIS.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_attr_destroy, pthread_attr_init — destroy and initialize the thread attributes object

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_init(pthread_attr_t *attr);
```

DESCRIPTION

The *pthread_attr_destroy*( ) function shall destroy a thread attributes object. An implementation may cause *pthread_attr_destroy*( ) to set *attr* to an implementation-defined invalid value. A destroyed *attr* attributes object can be reinitialized using *pthread_attr_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_attr_init*( ) function shall initialize a thread attributes object *attr* with the default value for all of the individual attributes used by a given implementation.

The resulting attributes object (possibly modified by setting individual attribute values) when used by *pthread_create*( ) defines the attributes of the thread created. A single attributes object can be used in multiple simultaneous calls to *pthread_create*( ). Results are undefined if *pthread_attr_init*( ) is called specifying an already initialized *attr* attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_destroy*( ) and *pthread_attr_init*( ) shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the thread attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to support probable future standardization in these areas without requiring that the function itself be changed.

Attributes objects provide clean isolation of the configurable aspects of threads. For example, ‘‘stack size’’ is an important attribute of a thread, but it cannot be expressed portably. When porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects can help by allowing the changes to be isolated in a single place, rather than being spread across every instance of thread creation.

Attributes objects can be used to set up ‘‘classes’ of threads with similar attributes; for example, ‘‘threads with large stacks and high priority’’ or ‘‘threads with minimal stacks’’. These classes can be defined in a single place and then referenced wherever threads need to be created. Changes to ‘‘class’’ decisions become straightforward, and detailed analysis of each *pthread_create*( ) call is not required.

The attributes objects are defined as opaque types as an aid to extensibility. If these objects had been specified as structures, adding new attributes would force recompilation of all multi-

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for improving performance. Argument validity can be checked once when attributes are set, rather than each time a thread is created. Implementations often need to cache kernel objects that are expensive to create. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Since assignment is not necessarily defined on a given opaque type, implementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attributes objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementation.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to the initialization routines (*pthread_create*( ), *pthread_mutex_init*( ), *pthread_cond_init*( )). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify implementation-defined flag values and associated behavior.

2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the **pthread_mutex_t** and **pthread_cond_t** objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**, application programmers need to know the location of each bit. If bits are set or read by encapsulation (that is, get and set functions), then the bitmask is merely an implementation of attributes objects as currently defined and should not be exposed to the programmer.

2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function parameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by get and set functions (making the bitmask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontiguous and can be allocated and released on demand.

The attribute mechanism has been designed in large measure for extensibility. Future extensions to the attribute mechanism or to any attributes object defined in this volume of IEEE Std 1003.1-2001 has to be done with care so as not to affect binary-compatibility.

Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc*( ), may have their size fixed at compile time. This means, for example, a *pthread_create*( ) in an implementation with extensions to **pthread_attr_t** cannot look beyond the area that the binary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as possibly version information, if extensions in different directions

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(possibly by different vendors) are to be accommodated.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_getstackaddr( ), *pthread_attr_getstacksize*( ), *pthread_attr_getdetachstate*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_destroy*( ) and *pthread_attr_init*( ) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already initialized thread attributes object is undefined.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_attr_getdetachstate, pthread_attr_setdetachstate — get and set the detachstate attribute

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_attr_getdetachstate(const pthread_attr_t *attr,
    int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

DESCRIPTION

The *detachstate* attribute controls whether the thread is created in a detached state. If the thread is created detached, then use of the ID of the newly created thread by the *pthread_detach*( ) or *pthread_join*( ) function is an error.

The *pthread_attr_getdetachstate*( ) and *pthread_attr_setdetachstate*( ) functions, respectively, shall get and set the *detachstate* attribute in the *attr* object.

For *pthread_attr_getdetachstate*( ), *detachstate* shall be set to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

For *pthread_attr_setdetachstate*( ), the application shall set *detachstate* to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

A value of PTHREAD_CREATE_DETACHED shall cause all threads created with *attr* to be in the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE shall cause all threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute shall be PTHREAD_CREATE_JOINABLE.

RETURN VALUE

Upon successful completion, *pthread_attr_getdetachstate*( ) and *pthread_attr_setdetachstate*( ) shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getdetachstate*( ) function stores the value of the *detachstate* attribute in *detachstate* if successful.

ERRORS

The *pthread_attr_setdetachstate*( ) function shall fail if:

[EINVAL] The value of *detachstate* was not valid

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getstackaddr*( ), *pthread_attr_getstacksize*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

31417 **CHANGE HISTORY**

31418 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31419 **Issue 6**

31420 The *pthread_attr_setdetachstate*( ) and *pthread_attr_getdetachstate*( ) functions are marked as part of
31421 the Threads option.

31422 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_getguardsize, pthread_attr_setguardsize — get and set the thread guardsize attribute

SYNOPSIS

XSI
```
#include <pthread.h>

int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
    size_t *restrict guardsize);
int pthread_attr_setguardsize(pthread_attr_t *attr,
    size_t guardsize);
```

DESCRIPTION

The *pthread_attr_getguardsize*( ) function shall get the *guardsize* attribute in the *attr* object. This attribute shall be returned in the *guardsize* parameter.

The *pthread_attr_setguardsize*( ) function shall set the *guardsize* attribute in the *attr* object. The new value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

The *guardsize* attribute controls the size of the guard area for the created thread's stack. The *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is created with guard protection, the implementation allocates extra memory at the overflow end of the stack as a buffer against stack overflow of the stack pointer. If an application overflows into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

A conforming implementation may round up the value contained in *guardsize* to a multiple of the configurable system variable {PAGESIZE} (see **<sys/mman.h>**). If an implementation rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread_attr_getguardsize*( ) specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous *pthread_attr_setguardsize*( ) function call.

The default value of the *guardsize* attribute is {PAGESIZE} bytes. The actual value of {PAGESIZE} is implementation-defined.

If the *stackaddr* or *stack* attribute has been set (that is, the caller is allocating and managing its own thread stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the implementation. It is the responsibility of the application to manage stack overflow along with stack allocation and management in this case.

RETURN VALUE

If successful, the *pthread_attr_getguardsize*( ) and *pthread_attr_setguardsize*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_getguardsize*( ) and *pthread_attr_setguardsize*( ) functions shall fail if:

[EINVAL] The attribute *attr* is invalid.

[EINVAL] The parameter *guardsize* is invalid.

These functions shall not return an error code of [EINTR].

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *guardsize* attribute is provided to the application for two reasons:

1. Overflow protection can potentially result in wasted system resources. An application that creates a large number of threads, and which knows its threads never overflow their stack, can save system resources by turning off guard areas.
2. When threads allocate large data structures on the stack, large guard areas may be needed to detect stack overflow.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<sys/mman.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the second error condition.

The **restrict** keyword is added to the *pthread_attr_getguardsize*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_getinheritsched, pthread_attr_setinheritsched — get and set the inheritsched attribute (**REALTIME THREADS**)

SYNOPSIS

THR TPS
```
#include <pthread.h>

int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
    int *restrict inheritsched);
int pthread_attr_setinheritsched(pthread_attr_t *attr,
    int inheritsched);
```

DESCRIPTION

The *pthread_attr_getinheritsched*( ), and *pthread_attr_setinheritsched*( ) functions, respectively, shall get and set the *inheritsched* attribute in the *attr* argument.

When the attributes objects are used by *pthread_create*( ), the *inheritsched* attribute determines how the other scheduling attributes of the created thread shall be set.

PTHREAD_INHERIT_SCHED
: Specifies that the thread scheduling attributes shall be inherited from the creating thread, and the scheduling attributes in this *attr* argument shall be ignored.

PTHREAD_EXPLICIT_SCHED
: Specifies that the thread scheduling attributes shall be set to the corresponding values from this attributes object.

The symbols PTHREAD_INHERIT_SCHED and PTHREAD_EXPLICIT_SCHED are defined in the **<pthread.h>** header.

The following thread scheduling attributes defined by IEEE Std 1003.1-2001 are affected by the *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and scheduling contention scope (*contentionscope*).

RETURN VALUE

If successful, the *pthread_attr_getinheritsched*( ) and *pthread_attr_setinheritsched*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setinheritsched*( ) function may fail if:

[EINVAL] The value of *inheritsched* is not valid.

[ENOTSUP] An attempt was made to set the attribute to an unsupported value.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*( ). Using these routines does not affect the current running thread.

RATIONALE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getschedparam*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_attr_getinheritsched*( ) and *pthread_attr_setinheritsched*( ) functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getinheritsched*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_getschedparam, pthread_attr_setschedparam — get and set the schedparam attribute

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
    struct sched_param *restrict param);
int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
    const struct sched_param *restrict param);
```

DESCRIPTION

The *pthread_attr_getschedparam*( ), and *pthread_attr_setschedparam*( ) functions, respectively, shall get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param* structure are defined in the **<sched.h>** header. For the SCHED_FIFO and SCHED_RR policies, the only required member of *param* is *sched_priority*.

TSP For the SCHED_SPORADIC policy, the required members of the *param* structure are *sched_priority*, *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The specified *sched_ss_repl_period* must be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail.

RETURN VALUE

If successful, the *pthread_attr_getschedparam*( ) and *pthread_attr_setschedparam*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setschedparam*( ) function may fail if:

[EINVAL] The value of *param* is not valid.

[ENOTSUP] An attempt was made to set the attribute to an unsupported value.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*( ). Using these routines does not affect the current running thread.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedpolicy*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<sched.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31585 **CHANGE HISTORY**

31586 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31587 **Issue 6**

31588 The *pthread_attr_getschedparam*( ) and *pthread_attr_setschedparam*( ) functions are marked as part
31589 of the Threads option.

31590 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

31591 The **restrict** keyword is added to the *pthread_attr_getschedparam*( ) and
31592 *pthread_attr_setschedparam*( ) prototypes for alignment with the ISO / IEC 9899: 1999 standard.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_getschedpolicy, pthread_attr_setschedpolicy — get and set the schedpolicy attribute (**REALTIME THREADS**)

SYNOPSIS

THR TPS
```
#include <pthread.h>

int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
    int *restrict policy);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

DESCRIPTION

The *pthread_attr_getschedpolicy*( ) and *pthread_attr_setschedpolicy*( ) functions, respectively, shall get and set the *schedpolicy* attribute in the *attr* argument.

The supported values of *policy* shall include SCHED_FIFO, SCHED_RR, and SCHED_OTHER, which are defined in the **<sched.h>** header. When threads executing with the scheduling policy SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on a mutex, they shall acquire the mutex in priority order when the mutex is unlocked.

TSP

RETURN VALUE

If successful, the *pthread_attr_getschedpolicy*( ) and *pthread_attr_setschedpolicy*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setschedpolicy*( ) function may fail if:

[EINVAL] The value of *policy* is not valid.

[ENOTSUP] An attempt was made to set the attribute to an unsupported value.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*( ). Using these routines does not affect the current running thread.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedparam*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 6

The *pthread_attr_getschedpolicy*( ) and *pthread_attr_setschedpolicy*( ) functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_attr_getschedpolicy*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_getscope, pthread_attr_setscope — get and set the contentionscope attribute (**REALTIME THREADS**)

SYNOPSIS

THR TPS
```
#include <pthread.h>

int pthread_attr_getscope(const pthread_attr_t *restrict attr,
    int *restrict contentionscope);
int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

DESCRIPTION

The *pthread_attr_getscope*( ) and *pthread_attr_setscope*( ) functions, respectively, shall get and set the *contentionscope* attribute in the *attr* object.

The *contentionscope* attribute may have the values PTHREAD_SCOPE_SYSTEM, signifying system scheduling contention scope, or PTHREAD_SCOPE_PROCESS, signifying process scheduling contention scope. The symbols PTHREAD_SCOPE_SYSTEM and PTHREAD_SCOPE_PROCESS are defined in the **<pthread.h>** header.

RETURN VALUE

If successful, the *pthread_attr_getscope*( ) and *pthread_attr_setscope*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setscope*( ) function may fail if:

[EINVAL] The value of *contentionscope* is not valid.

[ENOTSUP] An attempt was made to set the attribute to an unsupported value.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*( ). Using these routines does not affect the current running thread.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getschedparam*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

The *pthread_attr_getscope*( ) and *pthread_attr_setscope*( ) functions are marked as part of the Threads and Thread Execution Scheduling options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getscope*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_attr_getstack, pthread_attr_setstack — get and set stack attributes

SYNOPSIS

THR
```
#include <pthread.h>
```

TSA TSS
```
int pthread_attr_getstack(const pthread_attr_t *restrict attr,
    void **restrict stackaddr, size_t *restrict stacksize);
int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
    size_t stacksize);
```

DESCRIPTION

The *pthread_attr_getstack*( ) and *pthread_attr_setstack*( ) functions, respectively, shall get and set the thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

The stack attributes specify the area of storage to be used for the created thread's stack. The base (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be *stacksize* bytes. The *stacksize* shall be at least {PTHREAD_STACK_MIN}. The *stackaddr* shall be aligned appropriately to be used as a stack; for example, *pthread_attr_setstack*( ) may fail with [EINVAL] if (*stackaddr* & 0x7) is not 0. All pages within the stack described by *stackaddr* and *stacksize* shall be both readable and writable by the thread.

RETURN VALUE

Upon successful completion, these functions shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getstack*( ) function shall store the stack attribute values in *stackaddr* and *stacksize* if successful.

ERRORS

The *pthread_attr_setstack*( ) function shall fail if:

[EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds an implementation-defined limit.

The *pthread_attr_setstack*( ) function may fail if:

[EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or if (*stackaddr* + *stacksize*) lacks proper alignment.

[EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable and writable by the thread.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

These functions are appropriate for use by applications in an environment where the stack for a thread must be placed in some particular region of memory.

While it might seem that an application could detect stack overflow by providing a protected page outside the specified stack region, this cannot be done portably. Implementations are free to place the thread's initial stack pointer anywhere within the specified region to accommodate the machine's stack pointer behavior and allocation requirements. Furthermore, on some architectures, such as the IA-64, ''overflow'' might mean that two separate stack pointers allocated within the region will overlap somewhere in the middle of the region.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31732 **RATIONALE**

31733 None.

31734 **FUTURE DIRECTIONS**

31735 None.

31736 **SEE ALSO**

31737 *pthread_attr_init*( ), *pthread_attr_setdetachstate*( ), *pthread_attr_setstacksize*( ), *pthread_create*( ), the
31738 Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<pthread.h>**

31739 **CHANGE HISTORY**

31740 First released in Issue 6. Developed as an XSI extension and brought into the BASE by IEEE
31741 PASC Interpretation 1003.1 #101.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`--`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_getstackaddr, pthread_attr_setstackaddr — get and set the stackaddr attribute

SYNOPSIS

THR TSA
```
#include <pthread.h>
```

OB
```
int pthread_attr_getstackaddr(const pthread_attr_t *restrict attr,
    void **restrict stackaddr);
int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
```

DESCRIPTION

The *pthread_attr_getstackaddr*( ) and *pthread_attr_setstackaddr*( ) functions, respectively, shall get and set the thread creation *stackaddr* attribute in the *attr* object.

The *stackaddr* attribute specifies the location of storage to be used for the created thread's stack. The size of the storage shall be at least {PTHREAD_STACK_MIN}.

RETURN VALUE

Upon successful completion, *pthread_attr_getstackaddr*( ) and *pthread_attr_setstackaddr*( ) shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getstackaddr*( ) function stores the *stackaddr* attribute value in *stackaddr* if successful.

ERRORS

No errors are defined.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The specification of the *stackaddr* attribute presents several ambiguities that make portable use of these interfaces impossible. The description of the single address parameter as a ''stack'' does not specify a particular relationship between the address and the ''stack'' implied by that address. For example, the address may be taken as the low memory address of a buffer intended for use as a stack, or it may be taken as the address to be used as the initial stack pointer register value for the new thread. These two are not the same except for a machine on which the stack grows ''up'' from low memory to high, and on which a ''push'' operation first stores the value in memory and then increments the stack pointer register. Further, on a machine where the stack grows ''down'' from high memory to low, interpretation of the address as the ''low memory'' address requires a determination of the intended size of the stack. IEEE Std 1003.1-2001 has introduced the new interfaces *pthread_attr_setstack*( ) and *pthread_attr_getstack*( ) to resolve these ambiguities.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getdetachstate*( ), *pthread_attr_getstack*( ), *pthread_attr_getstacksize*( ), *pthread_attr_setstack*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<pthread.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31786 **CHANGE HISTORY**

31787 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31788 **Issue 6**

31789 The *pthread_attr_getstackaddr*( ) and *pthread_attr_setstackaddr*( ) functions are marked as part of
31790 the Threads and Thread Stack Address Attribute options.

31791 The **restrict** keyword is added to the *pthread_attr_getstackaddr*( ) prototype for alignment with the
31792 ISO / IEC 9899: 1999 standard.

31793 These functions are marked obsolescent.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_getstacksize, pthread_attr_setstacksize — get and set the stacksize attribute

SYNOPSIS

THR TSS
```
#include <pthread.h>

int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
    size_t *restrict stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

DESCRIPTION

The *pthread_attr_getstacksize*( ) and *pthread_attr_setstacksize*( ) functions, respectively, shall get and set the thread creation *stacksize* attribute in the *attr* object.

The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created threads stack.

RETURN VALUE

Upon successful completion, *pthread_attr_getstacksize*( ) and *pthread_attr_setstacksize*( ) shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getstacksize*( ) function stores the *stacksize* attribute value in *stacksize* if successful.

ERRORS

The *pthread_attr_setstacksize*( ) function shall fail if:

[EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds a system-imposed limit.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getstackaddr*( ), *pthread_attr_getdetachstate*( ), *pthread_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_attr_getstacksize*( ) and *pthread_attr_setstacksize*( ) functions are marked as part of the Threads and Thread Stack Size Attribute options.

The **restrict** keyword is added to the *pthread_attr_getstacksize*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from ‘‘Thread Stack Address Attribute’’ option to ‘‘Thread Stack Size Attribute’’ option.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

31838 **NAME**

31839 pthread_attr_init — initialize the thread attributes object

31840 **SYNOPSIS**

31841 THR
```
#include <pthread.h>
```

31842
```
int pthread_attr_init(pthread_attr_t *attr);
```

31843

31844 **DESCRIPTION**

31845 Refer to *pthread_attr_destroy*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31846 **NAME**

31847 pthread_attr_setdetachstate — set the detachstate attribute

31848 **SYNOPSIS**

31849 THR
```
#include <pthread.h>
```

31850
```
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

31851

31852 **DESCRIPTION**

31853 Refer to *pthread_attr_getdetachstate*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_setguardsize — set the thread guardsize attribute

SYNOPSIS

XSI

```
#include <pthread.h>

int pthread_attr_setguardsize(pthread_attr_t *attr,
    size_t guardsize);
```

DESCRIPTION

Refer to *pthread_attr_getguardsize*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31863 **NAME**

31864 pthread_attr_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)

31865 **SYNOPSIS**

31866 THR TPS
```
#include <pthread.h>

int pthread_attr_setinheritsched(pthread_attr_t *attr,
    int inheritsched);
```
31867
31868
31869

31870 **DESCRIPTION**

31871 Refer to *pthread_attr_getinheritsched*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

-·-`,,,`·-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_setschedparam — set the schedparam attribute

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
    const struct sched_param *restrict param);
```

DESCRIPTION

Refer to *pthread_attr_getschedparam*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31881 **NAME**

31882 pthread_attr_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)

31883 **SYNOPSIS**

31884 THR TPS
```
#include <pthread.h>

int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```
31885

31886

31887 **DESCRIPTION**

31888 Refer to *pthread_attr_getschedpolicy*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_attr_setscope — set the contentionscope attribute (**REALTIME THREADS**)

SYNOPSIS

THR TPS
```
#include <pthread.h>

int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

DESCRIPTION

Refer to *pthread_attr_getscope*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31897 **NAME**

31898 pthread_attr_setstack — set the stack attribute

31899 **SYNOPSIS**

31900 XSI
```
#include <pthread.h>

int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
    size_t stacksize);
```
31901
31902
31903

31904 **DESCRIPTION**

31905 Refer to *pthread_attr_getstack*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

31906 **NAME**

31907 pthread_attr_setstackaddr — set the stackaddr attribute

31908 **SYNOPSIS**

31909 THR TSA
```
#include <pthread.h>
```

31910 OB
```
int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
```

31911

31912 **DESCRIPTION**

31913 Refer to *pthread_attr_getstackaddr*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_attr_setstacksize — set the stacksize attribute

SYNOPSIS

THR TSS
```
#include <pthread.h>

int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

DESCRIPTION

Refer to *pthread_attr_getstacksize*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_barrier_destroy, pthread_barrier_init — destroy and initialize a barrier object (**ADVANCED REALTIME THREADS**)

SYNOPSIS

THR BAR
```
#include <pthread.h>

int pthread_barrier_destroy(pthread_barrier_t *barrier);
int pthread_barrier_init(pthread_barrier_t *restrict barrier,
    const pthread_barrierattr_t *restrict attr, unsigned count);
```

DESCRIPTION

The *pthread_barrier_destroy*( ) function shall destroy the barrier referenced by *barrier* and release any resources used by the barrier. The effect of subsequent use of the barrier is undefined until the barrier is reinitialized by another call to *pthread_barrier_init*( ). An implementation may use this function to set *barrier* to an invalid value. The results are undefined if *pthread_barrier_destroy*( ) is called when any thread is blocked on the barrier, or if this function is called with an uninitialized barrier.

The *pthread_barrier_init*( ) function shall allocate any resources required to use the barrier referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is NULL, the default barrier attributes shall be used; the effect is the same as passing the address of a default barrier attributes object. The results are undefined if *pthread_barrier_init*( ) is called when any thread is blocked on the barrier (that is, has not returned from the *pthread_barrier_wait*( ) call). The results are undefined if a barrier is used without first being initialized. The results are undefined if *pthread_barrier_init*( ) is called specifying an already initialized barrier.

The *count* argument specifies the number of threads that must call *pthread_barrier_wait*( ) before any of them successfully return from the call. The value specified by *count* must be greater than zero.

If the *pthread_barrier_init*( ) function fails, the barrier shall not be initialized and the contents of *barrier* are undefined.

Only the object referenced by *barrier* may be used for performing synchronization. The result of referring to copies of that object in calls to *pthread_barrier_destroy*( ) or *pthread_barrier_wait*( ) is undefined.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_barrier_destroy*( ) function may fail if:

[EBUSY] The implementation has detected an attempt to destroy a barrier while it is in use (for example, while being used in a *pthread_barrier_wait*( ) call) by another thread.

[EINVAL] The value specified by *barrier* is invalid.

The *pthread_barrier_init*( ) function shall fail if:

[EAGAIN] The system lacks the necessary resources to initialize another barrier.

[EINVAL] The value specified by *count* is equal to zero.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[ENOMEM] Insufficient memory exists to initialize the barrier.

The *pthread_barrier_init*( ) function may fail if:

[EBUSY] The implementation has detected an attempt to reinitialize a barrier while it is in use (for example, while being used in a *pthread_barrier_wait*( ) call) by another thread.

[EINVAL] The value specified by *attr* is invalid.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_barrier_destroy*( ) and *pthread_barrier_init*( ) functions are part of the Barriers option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_wait( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_barrier_wait — synchronize at a barrier (**ADVANCED REALTIME THREADS**)

SYNOPSIS

THR BAR
```
#include <pthread.h>

int pthread_barrier_wait(pthread_barrier_t *barrier);
```

DESCRIPTION

The *pthread_barrier_wait*( ) function shall synchronize participating threads at the barrier referenced by *barrier*. The calling thread shall block until the required number of threads have called *pthread_barrier_wait*( ) specifying the barrier.

When the required number of threads have called *pthread_barrier_wait*( ) specifying the barrier, the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall be reset to the state it had as a result of the most recent *pthread_barrier_init*( ) function that referenced it.

The constant PTHREAD_BARRIER_SERIAL_THREAD is defined in **<pthread.h>** and its value shall be distinct from any other value returned by *pthread_barrier_wait*( ).

The results are undefined if this function is called with an uninitialized barrier.

If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the required number of threads have not arrived at the barrier during the execution of the signal handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until the thread in the signal handler returns from it, it is unspecified whether other threads may proceed past the barrier once they have all reached it.

A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources shall be determined by the scheduling policy.

RETURN VALUE

Upon successful completion, the *pthread_barrier_wait*( ) function shall return PTHREAD_BARRIER_SERIAL_THREAD for a single (arbitrary) thread synchronized at the barrier and zero for each of the other threads. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_barrier_wait*( ) function may fail if:

[EINVAL] The value specified by *barrier* does not refer to an initialized barrier object.

This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

The *pthread_barrier_wait*( ) function is part of the Barriers option and need not be provided on all implementations.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32029 **RATIONALE**

32030 None.

32031 **FUTURE DIRECTIONS**

32032 None.

32033 **SEE ALSO**

32034 *pthread_barrier_destroy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

32035 **CHANGE HISTORY**

32036 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32037 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32038 **NAME**

32039 pthread_barrierattr_destroy, pthread_barrierattr_init — destroy and initialize the barrier
32040 attributes object (**ADVANCED REALTIME THREADS**)

32041 **SYNOPSIS**

32042 THR BAR
```
#include <pthread.h>

int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```
32045

32046 **DESCRIPTION**

32047 The *pthread_barrierattr_destroy*( ) function shall destroy a barrier attributes object. A destroyed
32048 *attr* attributes object can be reinitialized using *pthread_barrierattr_init*( ); the results of otherwise
32049 referencing the object after it has been destroyed are undefined. An implementation may cause
32050 *pthread_barrierattr_destroy*( ) to set the object referenced by *attr* to an invalid value.

32051 The *pthread_barrierattr_init*( ) function shall initialize a barrier attributes object *attr* with the
32052 default value for all of the attributes defined by the implementation.

32053 Results are undefined if *pthread_barrierattr_init*( ) is called specifying an already initialized *attr*
32054 attributes object.

32055 After a barrier attributes object has been used to initialize one or more barriers, any function
32056 affecting the attributes object (including destruction) shall not affect any previously initialized
32057 barrier.

32058 **RETURN VALUE**

32059 If successful, the *pthread_barrierattr_destroy*( ) and *pthread_barrierattr_init*( ) functions shall return
32060 zero; otherwise, an error number shall be returned to indicate the error.

32061 **ERRORS**

32062 The *pthread_barrierattr_destroy*( ) function may fail if:

32063 [EINVAL] The value specified by *attr* is invalid.

32064 The *pthread_barrierattr_init*( ) function shall fail if:

32065 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

32066 These functions shall not return an error code of [EINTR].

32067 **EXAMPLES**

32068 None.

32069 **APPLICATION USAGE**

32070 The *pthread_barrierattr_destroy*( ) and *pthread_barrierattr_init*( ) functions are part of the Barriers
32071 option and need not be provided on all implementations.

32072 **RATIONALE**

32073 None.

32074 **FUTURE DIRECTIONS**

32075 None.

32076 **SEE ALSO**

32077 *pthread_barrierattr_getpshared*( ), *pthread_barrierattr_setpshared*( ), the Base Definitions volume of
32078 IEEE Std 1003.1-2001, **<pthread.h>**.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32079 **CHANGE HISTORY**

32080 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32081 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_barrierattr_getpshared, pthread_barrierattr_setpshared — get and set the process-shared attribute of the barrier attributes object (**ADVANCED REALTIME THREADS**)

SYNOPSIS

THR
```
#include <pthread.h>
```

BAR TSH
```
int pthread_barrierattr_getpshared(const pthread_barrierattr_t *
    restrict attr, int *restrict pshared);
int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_barrierattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a barrier to be operated upon by any thread that has access to the memory where the barrier is allocated. If the *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the barrier shall only be operated upon by threads created within the same process as the thread that initialized the barrier; if threads of different processes attempt to operate on such a barrier, the behavior is undefined. The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE. Both constants PTHREAD_PROCESS_SHARED and PTHREAD_PROCESS_PRIVATE are defined in **<pthread.h>**.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

RETURN VALUE

If successful, the *pthread_barrierattr_getpshared*( ) function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_barrierattr_setpshared*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *pthread_barrierattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the *process-shared* attribute is not one of the legal values PTHREAD_PROCESS_SHARED or PTHREAD_PROCESS_PRIVATE.

These functions shall not return an error code of [EINTR].

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The *pthread_barrierattr_getpshared*( ) and *pthread_barrierattr_setpshared*( ) functions are part of the Barriers option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_destroy( ), *pthread_barrierattr_destroy*( ), *pthread_barrierattr_init*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_barrierattr_init — initialize the barrier attributes object (**ADVANCED REALTIME THREADS**)

SYNOPSIS

THR BAR

```
#include <pthread.h>

int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

DESCRIPTION

Refer to *pthread_barrierattr_destroy*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32142 **NAME**

32143 pthread_barrierattr_setpshared — set the process-shared attribute of the barrier attributes object
32144 (**ADVANCED REALTIME THREADS**)

32145 **SYNOPSIS**

32146 THR
```
#include <pthread.h>
```

32147 BAR TSH
```
int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
    int pshared);
```

32148
32149

32150 **DESCRIPTION**

32151 Refer to *pthread_barrierattr_getpshared*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32152 **NAME**

32153 pthread_cancel — cancel execution of a thread

32154 **SYNOPSIS**

32155 THR
```
#include <pthread.h>
```

32156
```
int pthread_cancel(pthread_t thread);
```

32157

32158 **DESCRIPTION**

32159 The *pthread_cancel*( ) function shall request that *thread* be canceled. The target thread's
32160 cancelability state and type determines when the cancellation takes effect. When the cancellation
32161 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last
32162 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called
32163 for *thread*. When the last destructor function returns, *thread* shall be terminated.

32164 The cancellation processing in the target thread shall run asynchronously with respect to the
32165 calling thread returning from *pthread_cancel*( ).

32166 **RETURN VALUE**

32167 If successful, the *pthread_cancel*( ) function shall return zero; otherwise, an error number shall be
32168 returned to indicate the error.

32169 **ERRORS**

32170 The *pthread_cancel*( ) function may fail if:

32171 [ESRCH] No thread could be found corresponding to that specified by the given thread
32172 ID.

32173 The *pthread_cancel*( ) function shall not return an error code of [EINTR].

32174 **EXAMPLES**

32175 None.

32176 **APPLICATION USAGE**

32177 None.

32178 **RATIONALE**

32179 Two alternative functions were considered for sending the cancellation notification to a thread.
32180 One would be to define a new SIGCANCEL signal that had the cancellation semantics when
32181 delivered; the other was to define the new *pthread_cancel*( ) function, which would trigger the
32182 cancellation semantics.

32183 The advantage of a new signal was that so much of the delivery criteria were identical to that
32184 used when trying to deliver a signal that making cancellation notification a signal was seen as
32185 consistent. Indeed, many implementations implement cancellation using a special signal. On the
32186 other hand, there would be no signal functions that could be used with this signal except
32187 *pthread_kill*( ), and the behavior of the delivered cancellation signal would be unlike any
32188 previously existing defined signal.

32189 The benefits of a special function include the recognition that this signal would be defined
32190 because of the similar delivery criteria and that this is the only common behavior between a
32191 cancellation request and a signal. In addition, the cancellation delivery mechanism does not
32192 have to be implemented as a signal. There are also strong, if not stronger, parallels with
32193 language exception mechanisms than with signals that are potentially obscured if the delivery
32194 mechanism is visibly closer to signals.

32195 In the end, it was considered that as there were so many exceptions to the use of the new signal
32196 with existing signals functions it would be misleading. A special function has resolved this

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

problem. This function was carefully defined so that an implementation wishing to provide the cancellation functions on top of signals could do so. The special function also means that implementations are not obliged to implement cancellation with signals.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_exit( ), *pthread_cond_timedwait*( ), *pthread_join*( ), *pthread_setcancelstate*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cancel*( ) function is marked as part of the Threads option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32209 **NAME**

32210 pthread_cleanup_pop, pthread_cleanup_push — establish cancellation handlers

32211 **SYNOPSIS**

32212 THR
```
#include <pthread.h>

void pthread_cleanup_pop(int execute);
void pthread_cleanup_push(void (*routine)(void*), void *arg);
```
32215

32216 **DESCRIPTION**

32217 The *pthread_cleanup_pop*( ) function shall remove the routine at the top of the calling thread's
32218 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).

32219 The *pthread_cleanup_push*( ) function shall push the specified cancellation cleanup handler *routine*
32220 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be
32221 popped from the cancellation cleanup stack and invoked with the argument *arg* when:

32222 • The thread exits (that is, calls *pthread_exit*( )).

32223 • The thread acts upon a cancellation request.

32224 • The thread calls *pthread_cleanup_pop*( ) with a non-zero *execute* argument.

32225 These functions may be implemented as macros. The application shall ensure that they appear
32226 as statements, and in pairs within the same lexical scope (that is, the *pthread_cleanup_push*( )
32227 macro may be thought to expand to a token list whose first token is '{' with
32228 *pthread_cleanup_pop*( ) expanding to a token list whose last token is the corresponding '}').

32229 The effect of calling *longjmp*( ) or *siglongjmp*( ) is undefined if there have been any calls to
32230 *pthread_cleanup_push*( ) or *pthread_cleanup_pop*( ) made without the matching call since the jump
32231 buffer was filled. The effect of calling *longjmp*( ) or *siglongjmp*( ) from inside a cancellation
32232 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation
32233 cleanup handler.

32234 **RETURN VALUE**

32235 The *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ) functions shall not return a value.

32236 **ERRORS**

32237 No errors are defined.

32238 These functions shall not return an error code of [EINTR].

32239 **EXAMPLES**

32240 The following is an example using thread primitives to implement a cancelable, writers-priority
32241 read-write lock:

```
32242 typedef struct {
32243     pthread_mutex_t lock;
32244     pthread_cond_t rcond,
32245         wcond;
32246     int lock_count; /* < 0 .. Held by writer. */
32247                     /* > 0 .. Held by lock_count readers. */
32248                     /* = 0 .. Held by nobody. */
32249     int waiting_writers; /* Count of waiting writers. */
32250 } rwlock;

32251 void
32252 waiting_reader_cleanup(void *arg)
32253 {
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
32254           rwlock *l;
32255           l = (rwlock *) arg;
32256           pthread_mutex_unlock(&l->lock);
32257       }
32258       void
32259       lock_for_read(rwlock *l)
32260       {
32261           pthread_mutex_lock(&l->lock);
32262           pthread_cleanup_push(waiting_reader_cleanup, l);
32263           while ((l->lock_count < 0) && (l->waiting_writers != 0))
32264               pthread_cond_wait(&l->rcond, &l->lock);
32265           l->lock_count++;
32266          /*
32267           * Note the pthread_cleanup_pop executes
32268           * waiting_reader_cleanup.
32269           */
32270           pthread_cleanup_pop(1);
32271       }
32272       void
32273       release_read_lock(rwlock *l)
32274       {
32275           pthread_mutex_lock(&l->lock);
32276           if (--l->lock_count == 0)
32277               pthread_cond_signal(&l->wcond);
32278           pthread_mutex_unlock(l);
32279       }
32280       void
32281       waiting_writer_cleanup(void *arg)
32282       {
32283           rwlock *l;
32284           l = (rwlock *) arg;
32285           if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
32286              /*
32287               * This only happens if we have been canceled.
32288               */
32289               pthread_cond_broadcast(&l->wcond);
32290       }
32291           pthread_mutex_unlock(&l->lock);
32292       }
32293       void
32294       lock_for_write(rwlock *l)
32295       {
32296           pthread_mutex_lock(&l->lock);
32297           l->waiting_writers++;
32298           pthread_cleanup_push(waiting_writer_cleanup, l);
32299           while (l->lock_count != 0)
32300               pthread_cond_wait(&l->wcond, &l->lock);
32301           l->lock_count = -1;
32302          /*
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
32303           * Note the pthread_cleanup_pop executes
32304           * waiting_writer_cleanup.
32305           */
32306          pthread_cleanup_pop(1);
32307      }
32308      void
32309      release_write_lock(rwlock *l)
32310      {
32311          pthread_mutex_lock(&l->lock);
32312          l->lock_count = 0;
32313          if (l->waiting_writers == 0)
32314              pthread_cond_broadcast(&l->rcond)
32315          else
32316              pthread_cond_signal(&l->wcond);
32317          pthread_mutex_unlock(&l->lock);
32318      }
32319      /*
32320       * This function is called to initialize the read/write lock.
32321       */
32322      void
32323      initialize_rwlock(rwlock *l)
32324      {
32325          pthread_mutex_init(&l->lock, pthread_mutexattr_default);
32326          pthread_cond_init(&l->wcond, pthread_condattr_default);
32327          pthread_cond_init(&l->rcond, pthread_condattr_default);
32328          l->lock_count = 0;
32329          l->waiting_writers = 0;
32330      }
32331      reader_thread()
32332      {
32333          lock_for_read(&lock);
32334          pthread_cleanup_push(release_read_lock, &lock);
32335         /*
32336          * Thread has read lock.
32337          */
32338          pthread_cleanup_pop(1);
32339      }
32340      writer_thread()
32341      {
32342          lock_for_write(&lock);
32343          pthread_cleanup_push(release_write_lock, &lock);
32344         /*
32345          * Thread has write lock.
32346          */
32347      pthread_cleanup_pop(1);
32348      }
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

APPLICATION USAGE

The two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ), can be thought of as left and right parentheses. They always need to be matched.

RATIONALE

The restriction that the two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push*( ) and *pthread_cleanup_pop*( ), have to appear in the same lexical scope allows for efficient macro or compiler implementations and efficient storage management. A sample implementation of these routines as macros might look like this:

```
#define pthread_cleanup_push(rtn,arg) { \
    struct _pthread_handler_rec __cleanup_handler, **__head; \
    __cleanup_handler.rtn = rtn; \
    __cleanup_handler.arg = arg; \
    (void) pthread_getspecific(_pthread_handler_key, &__head); \
    __cleanup_handler.next = *__head; \
    *__head = &__cleanup_handler;

#define pthread_cleanup_pop(ex) \
    *__head = __cleanup_handler.next; \
    if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
}
```

A more ambitious implementation of these routines might do even better by allowing the compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

This volume of IEEE Std 1003.1-2001 currently leaves unspecified the effect of calling *longjmp*( ) from a signal handler executing in a POSIX System Interfaces function. If an implementation wants to allow this and give the programmer reasonable behavior, the *longjmp*( ) function has to call all cancellation cleanup handlers that have been pushed but not popped since the time *setjmp*( ) was called.

Consider a multi-threaded function called by a thread that uses signals. If a signal were delivered to a signal handler during the operation of *qsort*( ) and that handler were to call *longjmp*( ) (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads created by the *qsort*( ) function would not be canceled. Instead, they would continue to execute and write into the argument array even though the array might have been popped off the stack.

Note that the specified cleanup handling mechanism is especially tied to the C language and, while the requirement for a uniform mechanism for expressing cleanup is language-independent, the mechanism used in other languages may be quite different. In addition, this mechanism is really only necessary due to the lack of a real exception mechanism in the C language, which would be the ideal solution.

There is no notion of a cancellation cleanup-safe function. If an application has no cancellation points in its signal handlers, blocks any signal whose handler may have cancellation points while calling async-unsafe functions, or disables cancellation while calling async-unsafe functions, all functions may be safely called from cancellation cleanup routines.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cancel( ), *pthread_setcancelstate*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cleanup_pop*() and *pthread_cleanup_push*() functions are marked as part of the Threads option.

The APPLICATION USAGE section is added.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_cond_broadcast, pthread_cond_signal — broadcast or signal a condition

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_signal(pthread_cond_t *cond);
```

DESCRIPTION

These functions shall unblock threads blocked on a condition variable.

The *pthread_cond_broadcast*( ) function shall unblock all threads currently blocked on the specified condition variable *cond*.

The *pthread_cond_signal*( ) function shall unblock at least one of the threads that are blocked on the specified condition variable *cond* (if any threads are blocked on *cond*).

If more than one thread is blocked on a condition variable, the scheduling policy shall determine the order in which threads are unblocked. When each thread unblocked as a result of a *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ) returns from its call to *pthread_cond_wait*( ) or *pthread_cond_timedwait*( ), the thread shall own the mutex with which it called *pthread_cond_wait*( ) or *pthread_cond_timedwait*( ). The thread(s) that are unblocked shall contend for the mutex according to the scheduling policy (if applicable), and as if each had called *pthread_mutex_lock*( ).

The *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ) functions may be called by a thread whether or not it currently owns the mutex that threads calling *pthread_cond_wait*( ) or *pthread_cond_timedwait*( ) have associated with the condition variable during their waits; however, if predictable scheduling behavior is required, then that mutex shall be locked by the thread calling *pthread_cond_broadcast*( ) or *pthread_cond_signal*( ).

The *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) functions shall have no effect if there are no threads currently blocked on *cond*.

RETURN VALUE

If successful, the *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) function may fail if:

[EINVAL] The value *cond* does not refer to an initialized condition variable.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_cond_broadcast*( ) function is used whenever the shared-variable state has been changed in a way that more than one thread can proceed with its task. Consider a single producer/multiple consumer problem, where the producer can insert multiple items on a list that is accessed one item at a time by the consumers. By calling the *pthread_cond_broadcast*( ) function, the producer would notify all consumers that might be waiting, and thereby the application would receive more throughput on a multi-processor. In addition, *pthread_cond_broadcast*( ) makes it easier to implement a read-write lock. The *pthread_cond_broadcast*( ) function is needed in order to wake up all waiting readers when a

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function to notify all clients of an impending transaction commit.

It is not safe to use the *pthread_cond_signal*() function in a signal handler that is invoked asynchronously. Even if it were safe, there would still be a race between the test of the Boolean *pthread_cond_wait*() that could not be efficiently eliminated.

Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling from code running in a signal handler.

RATIONALE

Multiple Awakenings by Condition Signal

On a multi-processor, it may be impossible for an implementation of *pthread_cond_signal*() to avoid the unblocking of more than one thread blocked on a condition variable. For example, consider the following partial implementation of *pthread_cond_wait*() and *pthread_cond_signal*(), executed by two threads in the order given. One thread is trying to wait on the condition variable, another is concurrently executing *pthread_cond_signal*(), while a third thread is already waiting.

```
pthread_cond_wait(mutex, cond):
    value = cond->value; /* 1 */
    pthread_mutex_unlock(mutex); /* 2 */
    pthread_mutex_lock(cond->mutex); /* 10 */
    if (value == cond->value) { /* 11 */
        me->next_cond = cond->waiter;
        cond->waiter = me;
        pthread_mutex_unlock(cond->mutex);
        unable_to_run(me);
    } else
        pthread_mutex_unlock(cond->mutex); /* 12 */
    pthread_mutex_lock(mutex); /* 13 */

pthread_cond_signal(cond):
    pthread_mutex_lock(cond->mutex); /* 3 */
    cond->value++; /* 4 */
    if (cond->waiter) { /* 5 */
        sleeper = cond->waiter; /* 6 */
        cond->waiter = sleeper->next_cond; /* 7 */
        able_to_run(sleeper); /* 8 */
    }
    pthread_mutex_unlock(cond->mutex); /* 9 */
```

The effect is that more than one thread can return from its call to *pthread_cond_wait*() or *pthread_cond_timedwait*() as a result of one call to *pthread_cond_signal*(). This effect is called ''spurious wakeup''. Note that the situation is self-correcting in that the number of threads that are so awakened is finite; for example, the next thread to call *pthread_cond_wait*() after the sequence of events above blocks.

While this problem could be resolved, the loss of efficiency for a fringe condition that occurs only rarely is unacceptable, especially given that one has to check the predicate associated with a condition variable anyway. Correcting this problem would unnecessarily reduce the degree of concurrency in this basic building block for all higher-level synchronization operations.

An added benefit of allowing spurious wakeups is that applications are forced to code a predicate-testing-loop around the condition wait. This also makes the application tolerate

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32494 superfluous condition broadcasts or signals on the same condition variable that may be coded in
32495 some other part of the application. The resulting applications are thus more robust. Therefore,
32496 IEEE Std 1003.1-2001 explicitly documents that spurious wakeups may occur.

32497 **FUTURE DIRECTIONS**

32498 None.

32499 **SEE ALSO**

32500 *pthread_cond_destroy*( ), *pthread_cond_timedwait*( ), the Base Definitions volume of
32501 IEEE Std 1003.1-2001, **<pthread.h>**

32502 **CHANGE HISTORY**

32503 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32504 **Issue 6**

32505 The *pthread_cond_broadcast*( ) and *pthread_cond_signal*( ) functions are marked as part of the
32506 Threads option.

32507 The APPLICATION USAGE section is added.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_cond_destroy, pthread_cond_init — destroy and initialize condition variables

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_init(pthread_cond_t *restrict cond,
    const pthread_condattr_t *restrict attr);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

DESCRIPTION

The *pthread_cond_destroy*( ) function shall destroy the given condition variable specified by *cond*; the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy*( ) to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can be reinitialized using *pthread_cond_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

It shall be safe to destroy an initialized condition variable upon which no threads are currently blocked. Attempting to destroy a condition variable upon which other threads are currently blocked results in undefined behavior.

The *pthread_cond_init*( ) function shall initialize the condition variable referenced by *cond* with attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be used; the effect is the same as passing the address of a default condition variable attributes object. Upon successful initialization, the state of the condition variable shall become initialized.

Only *cond* itself may be used for performing synchronization. The result of referring to copies of *cond* in calls to *pthread_cond_wait*( ), *pthread_cond_timedwait*( ), *pthread_cond_signal*( ), *pthread_cond_broadcast*( ), and *pthread_cond_destroy*( ) is undefined.

Attempting to initialize an already initialized condition variable results in undefined behavior.

In cases where default condition variable attributes are appropriate, the macro PTHREAD_COND_INITIALIZER can be used to initialize condition variables that are statically allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread_cond_init*( ) with parameter *attr* specified as NULL, except that no error checks are performed.

RETURN VALUE

If successful, the *pthread_cond_destroy*( ) and *pthread_cond_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

The [EBUSY] and [EINVAL] error checks, if implemented, shall act as if they were performed immediately at the beginning of processing for the function and caused an error return prior to modifying the state of the condition variable specified by *cond*.

ERRORS

The *pthread_cond_destroy*( ) function may fail if:

[EBUSY] The implementation has detected an attempt to destroy the object referenced by *cond* while it is referenced (for example, while being used in a *pthread_cond_wait*( ) or *pthread_cond_timedwait*( )) by another thread.

[EINVAL] The value specified by *cond* is invalid.

The *pthread_cond_init*( ) function shall fail if:

[EAGAIN] The system lacked the necessary resources (other than memory) to initialize another condition variable.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[ENOMEM] Insufficient memory exists to initialize the condition variable.

The *pthread_cond_init*( ) function may fail if:

[EBUSY] The implementation has detected an attempt to reinitialize the object referenced by *cond*, a previously initialized, but not yet destroyed, condition variable.

[EINVAL] The value specified by *attr* is invalid.

These functions shall not return an error code of [EINTR].

EXAMPLES

A condition variable can be destroyed immediately after all the threads that are blocked on it are awakened. For example, consider the following code:

```
struct list {
    pthread_mutex_t lm;
    ...
}

struct elt {
    key k;
    int busy;
    pthread_cond_t notbusy;
    ...
}

/* Find a list element and reserve it. */
struct elt *
list_find(struct list *lp, key k)
{
    struct elt *ep;

    pthread_mutex_lock(&lp->lm);
    while ((ep = find_elt(l, k) != NULL) && ep->busy)
        pthread_cond_wait(&ep->notbusy, &lp->lm);
    if (ep != NULL)
        ep->busy = 1;
    pthread_mutex_unlock(&lp->lm);
    return(ep);
}

delete_elt(struct list *lp, struct elt *ep)
{
    pthread_mutex_lock(&lp->lm);
    assert(ep->busy);
    ... remove ep from list ...
    ep->busy = 0;  /* Paranoid. */
(A) pthread_cond_broadcast(&ep->notbusy);
    pthread_mutex_unlock(&lp->lm);
(B) pthread_cond_destroy(&rp->notbusy);
    free(ep);
}
```

In this example, the condition variable and its list element may be freed (line B) immediately after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no other thread can touch the element to be deleted.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32600 **APPLICATION USAGE**

32601 None.

32602 **RATIONALE**

32603 See *pthread_mutex_init*( ); a similar rationale applies to condition variables.

32604 **FUTURE DIRECTIONS**

32605 None.

32606 **SEE ALSO**

32607 *pthread_cond_broadcast*( ), *pthread_cond_signal*( ), *pthread_cond_timedwait*( ), the Base Definitions
32608 volume of IEEE Std 1003.1-2001, **<pthread.h>**

32609 **CHANGE HISTORY**

32610 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32611 **Issue 6**

32612 The *pthread_cond_destroy*( ) and *pthread_cond_init*( ) functions are marked as part of the Threads
32613 option.

32614 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

32615 The **restrict** keyword is added to the *pthread_cond_init*( ) prototype for alignment with the
32616 ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32617 **NAME**

32618 pthread_cond_signal — signal a condition

32619 **SYNOPSIS**

32620 THR
```
#include <pthread.h>
```

32621
```
int pthread_cond_signal(pthread_cond_t *cond);
```

32622

32623 **DESCRIPTION**

32624 Refer to *pthread_cond_broadcast*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32625 **NAME**

32626 pthread_cond_timedwait, pthread_cond_wait — wait on a condition

32627 **SYNOPSIS**

32628 THR
```
#include <pthread.h>

int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

32634

32635 **DESCRIPTION**

32636 The *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) functions shall block on a condition
32637 variable. They shall be called with *mutex* locked by the calling thread or undefined behavior
32638 results.

32639 These functions atomically release *mutex* and cause the calling thread to block on the condition
32640 variable *cond*; atomically here means ''atomically with respect to access by another thread to the
32641 mutex and then the condition variable''. That is, if another thread is able to acquire the mutex
32642 after the about-to-block thread has released it, then a subsequent call to *pthread_cond_broadcast*( )
32643 or *pthread_cond_signal*( ) in that thread shall behave as if it were issued after the about-to-block
32644 thread has blocked.

32645 Upon successful return, the mutex shall have been locked and shall be owned by the calling
32646 thread.

32647 When using condition variables there is always a Boolean predicate involving shared variables
32648 associated with each condition wait that is true if the thread should proceed. Spurious wakeups
32649 from the *pthread_cond_timedwait*( ) or *pthread_cond_wait*( ) functions may occur. Since the return
32650 from *pthread_cond_timedwait*( ) or *pthread_cond_wait*( ) does not imply anything about the value
32651 of this predicate, the predicate should be re-evaluated upon such return.

32652 The effect of using more than one mutex for concurrent *pthread_cond_timedwait*( ) or
32653 *pthread_cond_wait*( ) operations on the same condition variable is undefined; that is, a condition
32654 variable becomes bound to a unique mutex when a thread waits on the condition variable, and
32655 this (dynamic) binding shall end when the wait returns.

32656 A condition wait (whether timed or not) is a cancellation point. When the cancelability enable
32657 state of a thread is set to PTHREAD_CANCEL_DEFERRED, a side effect of acting upon a
32658 cancellation request while in a condition wait is that the mutex is (in effect) re-acquired before
32659 calling the first cancellation cleanup handler. The effect is as if the thread were unblocked,
32660 allowed to execute up to the point of returning from the call to *pthread_cond_timedwait*( ) or
32661 *pthread_cond_wait*( ), but at that point notices the cancellation request and instead of returning to
32662 the caller of *pthread_cond_timedwait*( ) or *pthread_cond_wait*( ), starts the thread cancellation
32663 activities, which includes calling cancellation cleanup handlers.

32664 A thread that has been unblocked because it has been canceled while blocked in a call to
32665 *pthread_cond_timedwait*( ) or *pthread_cond_wait*( ) shall not consume any condition signal that
32666 may be directed concurrently at the condition variable if there are other threads blocked on the
32667 condition variable.

32668 The *pthread_cond_timedwait*( ) function shall be equivalent to *pthread_cond_wait*( ), except that an
32669 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or
32670 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time
32671 specified by *abstime* has already been passed at the time of the call.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32672 CS If the Clock Selection option is supported, the condition variable shall have a clock attribute
32673 which specifies the clock that shall be used to measure the time specified by the *abstime*
32674 argument. When such timeouts occur, *pthread_cond_timedwait*( ) shall nonetheless release and
32675 re-acquire the mutex referenced by *mutex*. The *pthread_cond_timedwait*( ) function is also a
32676 cancellation point.

32677 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal
32678 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it
32679 shall return zero due to spurious wakeup.

32680 **RETURN VALUE**

32681 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed
32682 immediately at the beginning of processing for the function and shall cause an error return, in
32683 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable
32684 specified by *cond*.

32685 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall
32686 be returned to indicate the error.

32687 **ERRORS**

32688 The *pthread_cond_timedwait*( ) function shall fail if:

32689 [ETIMEDOUT] The time specified by *abstime* to *pthread_cond_timedwait*( ) has passed.

32690 The *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) functions may fail if:

32691 [EINVAL] The value specified by *cond*, *mutex*, or *abstime* is invalid.

32692 [EINVAL] Different mutexes were supplied for concurrent *pthread_cond_timedwait*( ) or
32693 *pthread_cond_wait*( ) operations on the same condition variable.

32694 [EPERM] The mutex was not owned by the current thread at the time of the call.

32695 These functions shall not return an error code of [EINTR].

32696 **EXAMPLES**

32697 None.

32698 **APPLICATION USAGE**

32699 None.

32700 **RATIONALE**

32701 **Condition Wait Semantics**

32702 It is important to note that when *pthread_cond_wait*( ) and *pthread_cond_timedwait*( ) return
32703 without error, the associated predicate may still be false. Similarly, when
32704 *pthread_cond_timedwait*( ) returns with the timeout error, the associated predicate may be true
32705 due to an unavoidable race between the expiration of the timeout and the predicate state change.

32706 Some implementations, particularly on a multi-processor, may sometimes cause multiple
32707 threads to wake up when the condition variable is signaled simultaneously on different
32708 processors.

32709 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate
32710 associated with the condition wait to determine whether it can safely proceed, should wait
32711 again, or should declare a timeout. A return from the wait does not imply that the associated
32712 predicate is either true or false.

32713 It is thus recommended that a condition wait be enclosed in the equivalent of a ‘‘while loop’’
32714 that checks the predicate.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Timed Wait Semantics

An absolute time measure was chosen for specifying the timeout parameter for two reasons. First, a relative time measure can be easily implemented on top of a function that specifies absolute time, but there is a race condition associated with specifying an absolute timeout on top of a function that specifies relative timeouts. For example, assume that *clock_gettime*( ) returns the current time and *cond_relative_timed_wait*( ) uses relative timeouts:

```
clock_gettime(CLOCK_REALTIME, &now)
reltime = sleep_til_this_absolute_time -now;
cond_relative_timed_wait(c, m, &reltime);
```

If the thread is preempted between the first statement and the last statement, the thread blocks for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout also need not be recomputed if it is used multiple times in a loop, such as that enclosing a condition wait.

For cases when the system clock is advanced discontinuously by an operator, it is expected that implementations process any timed wait expiring at an intervening time as if that time had actually occurred.

Cancellation and Condition Wait

A condition wait, whether timed or not, is a cancellation point. That is, the functions *pthread_cond_wait*( ) or *pthread_cond_timedwait*( ) are points where a pending (or concurrent) cancellation request is noticed. The reason for this is that an indefinite wait is possible at these points—whatever event is being waited for, even if the program is totally correct, might never occur; for example, some input data being awaited might never be sent. By making condition wait a cancellation point, the thread can be canceled and perform its cancellation cleanup handler even though it may be stuck in some indefinite wait.

A side effect of acting on a cancellation request while a thread is blocked on a condition variable is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in order to ensure that the cancellation cleanup handler is executed in the same state as the critical code that lies both before and after the call to the condition wait function. This rule is also required when interfacing to POSIX threads from languages, such as Ada or C++, which may choose to map cancellation onto a language exception; this rule ensures that each exception handler guarding a critical section can always safely depend upon the fact that the associated mutex has already been locked regardless of exactly where within the critical section the exception was raised. Without this rule, there would not be a uniform rule that exception handlers could follow regarding the lock, and so coding would become very cumbersome.

Therefore, since *some* statement has to be made regarding the state of the lock when a cancellation is delivered during a wait, a definition has been chosen that makes application coding most convenient and error free.

When acting on a cancellation request while a thread is blocked on a condition variable, the implementation is required to ensure that the thread does not consume any condition signals directed at that condition variable if there are any other threads waiting on that condition variable. This rule is specified in order to avoid deadlock conditions that could occur if these two independent requests (one acting on a thread and the other acting on the condition variable) were not processed independently.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32758 **Performance of Mutexes and Condition Variables**

32759 Mutexes are expected to be locked only for a few instructions. This practice is almost
32760 automatically enforced by the desire of programmers to avoid long serial regions of execution
32761 (which would reduce total effective parallelism).

32762 When using mutexes and condition variables, one tries to ensure that the usual case is to lock the
32763 mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a
32764 relatively rare situation. For example, when implementing a read-write lock, code that acquires a
32765 read-lock typically needs only to increment the count of readers (under mutual-exclusion) and
32766 return. The calling thread would actually wait on the condition variable only when there is
32767 already an active writer. So the efficiency of a synchronization operation is bounded by the cost
32768 of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context
32769 switch.

32770 This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be
32771 at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable
32772 is important. The cost of waiting on a condition variable should be little more than the minimal
32773 cost for a context switch plus the time to unlock and lock the mutex.

32774 **Features of Mutexes and Condition Variables**

32775 It had been suggested that the mutex acquisition and release be decoupled from condition wait.
32776 This was rejected because it is the combined nature of the operation that, in fact, facilitates
32777 realtime implementations. Those implementations can atomically move a high-priority thread
32778 between the condition variable and the mutex in a manner that is transparent to the caller. This
32779 can prevent extra context switches and provide more deterministic acquisition of a mutex when
32780 the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the
32781 scheduling discipline. Furthermore, the current condition wait operation matches existing
32782 practice.

32783 **Scheduling Behavior of Mutexes and Condition Variables**

32784 Synchronization primitives that attempt to interfere with scheduling policy by specifying an
32785 ordering rule are considered undesirable. Threads waiting on mutexes and condition variables
32786 are selected to proceed in an order dependent upon the scheduling policy rather than in some
32787 fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which
32788 thread(s) are awakened and allowed to proceed.

32789 **Timed Condition Wait**

32790 The *pthread_cond_timedwait*( ) function allows an application to give up waiting for a particular
32791 condition after a given amount of time. An example of its use follows:

```
32792 (void) pthread_mutex_lock(&t.mn);
32793         t.waiters++;
32794     clock_gettime(CLOCK_REALTIME, &ts);
32795     ts.tv_sec += 5;
32796     rc = 0;
32797     while (! mypredicate(&t) && rc == 0)
32798         rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
32799     t.waiters--;
32800     if (rc == 0) setmystate(&t);
32801 (void) pthread_mutex_unlock(&t.mn);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

By making the timeout parameter absolute, it does not need to be recomputed each time the program checks its blocking predicate. If the timeout was relative, it would have to be recomputed before each call. This would be especially difficult since such code would need to take into account the possibility of extra wakeups that result from extra broadcasts or signals on the condition variable that occur before either the predicate is true or the timeout is due.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_signal( ), *pthread_cond_broadcast*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) functions are marked as part of the Threads option.

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the *pthread_cond_wait*( ) function.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for the Clock Selection option.

The ERRORS section has an additional case for [EPERM] in response to IEEE PASC Interpretation 1003.1c #28.

The **restrict** keyword is added to the *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_condattr_destroy, pthread_condattr_init — destroy and initialize the condition variable attributes object

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_condattr_destroy(pthread_condattr_t *attr);
int pthread_condattr_init(pthread_condattr_t *attr);
```

DESCRIPTION

The *pthread_condattr_destroy*( ) function shall destroy a condition variable attributes object; the object becomes, in effect, uninitialized. An implementation may cause *pthread_condattr_destroy*( ) to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be reinitialized using *pthread_condattr_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_condattr_init*( ) function shall initialize a condition variable attributes object *attr* with the default value for all of the attributes defined by the implementation.

Results are undefined if *pthread_condattr_init*( ) is called specifying an already initialized *attr* attributes object.

After a condition variable attributes object has been used to initialize one or more condition variables, any function affecting the attributes object (including destruction) shall not affect any previously initialized condition variables.

This volume of IEEE Std 1003.1-2001 requires two attributes, the *clock* attribute and the *process-shared* attribute.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

RETURN VALUE

If successful, the *pthread_condattr_destroy*( ) and *pthread_condattr_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_destroy*( ) function may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *pthread_condattr_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See *pthread_attr_init*( ) and *pthread_mutex_init*( ).

A *process-shared* attribute has been defined for condition variables for the same reason it has been defined for mutexes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_cond_destroy*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_mutex_destroy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_condattr_destroy*( ) and *pthread_condattr_init*( ) functions are marked as part of the Threads option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_condattr_getclock, pthread_condattr_setclock — get and set the clock selection condition variable attribute (**ADVANCED REALTIME**)

SYNOPSIS

THR CS

```
#include <pthread.h>

int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
    clockid_t *restrict clock_id);
int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```

DESCRIPTION

The *pthread_condattr_getclock*( ) function shall obtain the value of the *clock* attribute from the attributes object referenced by *attr*. The *pthread_condattr_setclock*( ) function shall set the *clock* attribute in an initialized attributes object referenced by *attr*. If *pthread_condattr_setclock*( ) is called with a *clock_id* argument that refers to a CPU-time clock, the call shall fail.

The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of *pthread_cond_timedwait*( ). The default value of the *clock* attribute shall refer to the system clock.

RETURN VALUE

If successful, the *pthread_condattr_getclock*( ) function shall return zero and store the value of the clock attribute of *attr* into the object referenced by the *clock_id* argument. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_condattr_setclock*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *pthread_condattr_setclock*( ) function may fail if:

[EINVAL] The value specified by *clock_id* does not refer to a known clock, or is a CPU-time clock.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_cond_timedwait*( ), *pthread_condattr_destroy*( ), *pthread_condattr_getpshared*( ) (on page 1045),1 *pthread_condattr_init*( ), *pthread_condattr_setpshared*( ) (on page 1049),1 *pthread_create*( ), *pthread_mutex_init*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32920 **CHANGE HISTORY**

32921 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_condattr_getpshared, pthread_condattr_setpshared — get and set the process-shared condition variable attributes

SYNOPSIS

THR TSH
```
#include <pthread.h>

int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
    int *restrict pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_condattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*. The *pthread_condattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a condition variable to be operated upon by any thread that has access to the memory where the condition variable is allocated, even if the condition variable is allocated in memory that is shared by multiple processes. If the *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the condition variable shall only be operated upon by threads created within the same process as the thread that initialized the condition variable; if threads of differing processes attempt to operate on such a condition variable, the behavior is undefined. The default value of the attribute is PTHREAD_PROCESS_PRIVATE.

RETURN VALUE

If successful, the *pthread_condattr_setpshared*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_condattr_getpshared*( ) function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_getpshared*( ) and *pthread_condattr_setpshared*( ) functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *pthread_condattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_cond_destroy*( ), *pthread_condattr_destroy*( ), *pthread_mutex_destroy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_condattr_getpshared*( ) and *pthread_condattr_setpshared*( ) functions are marked as part of the Threads and Thread Process-Shared Synchronization options.

The **restrict** keyword is added to the *pthread_condattr_getpshared*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32975 **NAME**

32976 pthread_condattr_init — initialize the condition variable attributes object

32977 **SYNOPSIS**

32978 THR
```
#include <pthread.h>

int pthread_condattr_init(pthread_condattr_t *attr);
```
32979

32980

32981 **DESCRIPTION**

32982 Refer to *pthread_condattr_destroy*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32983 **NAME**

32984 pthread_condattr_setclock — set the clock selection condition variable attribute

32985 **SYNOPSIS**

32986 THR CS
```
#include <pthread.h>

int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```
32989

32990 **DESCRIPTION**

32991 Refer to *pthread_condattr_getclock*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32992 **NAME**

32993 pthread_condattr_setpshared — set the process-shared condition variable attribute

32994 **SYNOPSIS**

32995 THR TSH
```
#include <pthread.h>

int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```
32998

32999 **DESCRIPTION**

33000 Refer to *pthread_condattr_getpshared*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33001 **NAME**

33002 pthread_create — thread creation

33003 **SYNOPSIS**

33004 THR
```
#include <pthread.h>

int pthread_create(pthread_t *restrict thread,
    const pthread_attr_t *restrict attr,
    void *(*start_routine)(void*), void *restrict arg);
```
33005

33006

33007

33008

33009 **DESCRIPTION**

33010 The *pthread_create*( ) function shall create a new thread, with attributes specified by *attr*, within a
33011 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are
33012 modified later, the thread's attributes shall not be affected. Upon successful completion,
33013 *pthread_create*( ) shall store the ID of the created thread in the location referenced by *thread*.

33014 The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine*
33015 returns, the effect shall be as if there was an implicit call to *pthread_exit*( ) using the return value
33016 of *start_routine* as the exit status. Note that the thread in which *main*( ) was originally invoked
33017 differs from this. When it returns from *main*( ), the effect shall be as if there was an implicit call
33018 to *exit*( ) using the return value of *main*( ) as the exit status.

33019 The signal state of the new thread shall be initialized as follows:

33020 - The signal mask shall be inherited from the creating thread.

33021 - The set of signals pending for the new thread shall be empty.

33022 XSI The alternate stack shall not be inherited.

33023 The floating-point environment shall be inherited from the creating thread.

33024 If *pthread_create*( ) fails, no new thread is created and the contents of the location referenced by
33025 *thread* are undefined.

33026 TCT If _POSIX_THREAD_CPUTIME is defined, the new thread shall have a CPU-time clock
33027 accessible, and the initial value of this clock shall be set to zero.

33028 **RETURN VALUE**

33029 If successful, the *pthread_create*( ) function shall return zero; otherwise, an error number shall be
33030 returned to indicate the error.

33031 **ERRORS**

33032 The *pthread_create*( ) function shall fail if:

33033 [EAGAIN] The system lacked the necessary resources to create another thread, or the
33034 system-imposed limit on the total number of threads in a process
33035 {PTHREAD_THREADS_MAX} would be exceeded.

33036 [EINVAL] The value specified by *attr* is invalid.

33037 [EPERM] The caller does not have appropriate permission to set the required
33038 scheduling parameters or scheduling policy.

33039 The *pthread_create*( ) function shall not return an error code of [EINTR].

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A suggested alternative to *pthread_create*() would be to define two separate operations: create and start. Some applications would find such behavior more natural. Ada, in particular, separates the ''creation'' of a task from its ''activation''.

Splitting the operation was rejected by the standard developers for many reasons:

- The number of calls required to start a thread would increase from one to two and thus place an additional burden on applications that do not require the additional synchronization. The second call, however, could be avoided by the additional complication of a start-up state attribute.
- An extra state would be introduced: ''created but not started''. This would require the standard to specify the behavior of the thread operations when the target has not yet started executing.
- For those applications that require such behavior, it is possible to simulate the two separate steps with the facilities that are currently provided. The *start_routine*() can synchronize by waiting on a condition variable that is signaled by the start operation.

An Ada implementor can choose to create the thread at either of two points in the Ada program: when the task object is created, or when the task is activated (generally at a ''begin''). If the first approach is adopted, the *start_routine*() needs to wait on a condition variable to receive the order to begin ''activation''. The second approach requires no such condition variable or extra synchronization. In either approach, a separate Ada task control block would need to be created when the task object is created to hold rendezvous queues, and so on.

An extension of the preceding model would be to allow the state of the thread to be modified between the create and start. This would allow the thread attributes object to be eliminated. This has been rejected because:

- All state in the thread attributes object has to be able to be set for the thread. This would require the definition of functions to modify thread attributes. There would be no reduction in the number of function calls required to set up the thread. In fact, for an application that creates all threads using identical attributes, the number of function calls required to set up the threads would be dramatically increased. Use of a thread attributes object permits the application to make one set of attribute setting function calls. Otherwise, the set of attribute setting function calls needs to be made for each thread creation.
- Depending on the implementation architecture, functions to set thread state would require kernel calls, or for other implementation reasons would not be able to be implemented as macros, thereby increasing the cost of thread creation.
- The ability for applications to segregate threads by class would be lost.

Another suggested alternative uses a model similar to that for process creation, such as ''thread fork''. The fork semantics would provide more flexibility and the ''create'' function can be implemented simply by doing a thread fork followed immediately by a call to the desired ''start routine'' for the thread. This alternative has these problems:

- For many implementations, the entire stack of the calling thread would need to be duplicated, since in many architectures there is no way to determine the size of the calling frame.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- Efficiency is reduced since at least some part of the stack has to be copied, even though in most cases the thread never needs the copied context, since it merely calls the desired start routine.

FUTURE DIRECTIONS

None.

SEE ALSO

fork( ), *pthread_exit*( ), *pthread_join*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_create*( ) function is marked as part of the Threads option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EPERM] mandatory error condition is added.

The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_create*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION is updated to make it explicit that the floating-point environment is inherited from the creating thread.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the alternate stack is not inherited.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_detach — detach a thread

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

DESCRIPTION

The *pthread_detach*( ) function shall indicate to the implementation that storage for the thread *thread* can be reclaimed when that thread terminates. If *thread* has not terminated, *pthread_detach*( ) shall not cause it to terminate. The effect of multiple *pthread_detach*( ) calls on the same target thread is unspecified.

RETURN VALUE

If the call succeeds, *pthread_detach*( ) shall return 0; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_detach*( ) function shall fail if:

[EINVAL] The implementation has detected that the value specified by *thread* does not refer to a joinable thread.

[ESRCH] No thread could be found corresponding to that specified by the given thread ID.

The *pthread_detach*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pthread_join*( ) or *pthread_detach*( ) functions should eventually be called for every thread that is created so that storage associated with the thread may be reclaimed.

It has been suggested that a ''detach'' function is not necessary; the *detachstate* thread creation attribute is sufficient, since a thread need never be dynamically detached. However, need arises in at least two cases:

1. In a cancellation handler for a *pthread_join*( ) it is nearly essential to have a *pthread_detach*( ) function in order to detach the thread on which *pthread_join*( ) was waiting. Without it, it would be necessary to have the handler do another *pthread_join*( ) to attempt to detach the thread, which would both delay the cancellation processing for an unbounded period and introduce a new call to *pthread_join*( ), which might itself need a cancellation handler. A dynamic detach is nearly essential in this case.
2. In order to detach the ''initial thread'' (as may be desirable in processes that set up server threads).

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33149 **SEE ALSO**

33150 *pthread_join*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

33151 **CHANGE HISTORY**

33152 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33153 **Issue 6**

33154 The *pthread_detach*( ) function is marked as part of the Threads option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

33155 **NAME**

33156 pthread_equal — compare thread IDs

33157 **SYNOPSIS**

33158 THR
```
#include <pthread.h>
```

33159
```
int pthread_equal(pthread_t t1, pthread_t t2);
```

33160

33161 **DESCRIPTION**

33162 This function shall compare the thread IDs *t1* and *t2*.

33163 **RETURN VALUE**

33164 The *pthread_equal*( ) function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero
33165 shall be returned.

33166 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.

33167 **ERRORS**

33168 No errors are defined.

33169 The *pthread_equal*( ) function shall not return an error code of [EINTR].

33170 **EXAMPLES**

33171 None.

33172 **APPLICATION USAGE**

33173 None.

33174 **RATIONALE**

33175 Implementations may choose to define a thread ID as a structure. This allows additional
33176 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence
33177 number that allows detection of ‘‘dangling IDs’’ (copies of a thread ID that has been detached).
33178 Since the C language does not support comparison on structure types, the *pthread_equal*( )
33179 function is provided to compare thread IDs.

33180 **FUTURE DIRECTIONS**

33181 None.

33182 **SEE ALSO**

33183 *pthread_create*( ), *pthread_self*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

33184 **CHANGE HISTORY**

33185 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33186 **Issue 6**

33187 The *pthread_equal*( ) function is marked as part of the Threads option.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33188 **NAME**

33189 pthread_exit — thread termination

33190 **SYNOPSIS**

33191 THR
```
#include <pthread.h>
```

33192
```
void pthread_exit(void *value_ptr);
```

33193

33194 **DESCRIPTION**

33195 The *pthread_exit*( ) function shall terminate the calling thread and make the value *value_ptr*
33196 available to any successful join with the terminating thread. Any cancellation cleanup handlers |
33197 that have been pushed and not yet popped shall be popped in the reverse order that they were |
33198 pushed and then executed. After all cancellation cleanup handlers have been executed, if the |
33199 thread has any thread-specific data, appropriate destructor functions shall be called in an
33200 unspecified order. Thread termination does not release any application visible process resources,
33201 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level
33202 cleanup actions, including, but not limited to, calling any *atexit*( ) routines that may exist.

33203 An implicit call to *pthread_exit*( ) is made when a thread other than the thread in which *main*( )
33204 was first invoked returns from the start routine that was used to create it. The function's return
33205 value shall serve as the thread's exit status.

33206 The behavior of *pthread_exit*( ) is undefined if called from a cancellation cleanup handler or |
33207 destructor function that was invoked as a result of either an implicit or explicit call to
33208 *pthread_exit*( ).

33209 After a thread has terminated, the result of access to local (auto) variables of the thread is
33210 undefined. Thus, references to local variables of the exiting thread should not be used for the
33211 *pthread_exit*( ) *value_ptr* parameter value.

33212 The process shall exit with an exit status of 0 after the last thread has been terminated. The
33213 behavior shall be as if the implementation called *exit*( ) with a zero argument at thread
33214 termination time.

33215 **RETURN VALUE**

33216 The *pthread_exit*( ) function cannot return to its caller.

33217 **ERRORS**

33218 No errors are defined.

33219 **EXAMPLES**

33220 None.

33221 **APPLICATION USAGE**

33222 None.

33223 **RATIONALE**

33224 The normal mechanism by which a thread terminates is to return from the routine that was
33225 specified in the *pthread_create*( ) call that started it. The *pthread_exit*( ) function provides the
33226 capability for a thread to terminate without requiring a return from the start routine of that
33227 thread, thereby providing a function analogous to *exit*( ).

33228 Regardless of the method of thread termination, any cancellation cleanup handlers that have |
33229 been pushed and not yet popped are executed, and the destructors for any existing thread- |
33230 specific data are executed. This volume of IEEE Std 1003.1-2001 requires that cancellation |
33231 cleanup handlers be popped and called in order. After all cancellation cleanup handlers have |
33232 been executed, thread-specific data destructors are called, in an unspecified order, for each item
33233 of thread-specific data that exists in the thread. This ordering is necessary because cancellation |

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

cleanup handlers may rely on thread-specific data.

As the meaning of the status is determined by the application (except when the thread has been canceled, in which case it is PTHREAD_CANCELED), the implementation has no idea what an illegal status value is, which is why no address error checking is done.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *pthread_create*( ), *pthread_join*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_exit*( ) function is marked as part of the Threads option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_getconcurrency, pthread_setconcurrency — get and set the level of concurrency

SYNOPSIS

XSI

```
#include <pthread.h>

int pthread_getconcurrency(void);
int pthread_setconcurrency(int new_level);
```

DESCRIPTION

Unbound threads in a process may or may not be required to be simultaneously active. By default, the threads implementation ensures that a sufficient number of threads are active so that the process can continue to make progress. While this conserves system resources, it may not produce the most effective level of concurrency.

The *pthread_setconcurrency*( ) function allows an application to inform the threads implementation of its desired concurrency level, *new_level*. The actual level of concurrency provided by the implementation as a result of this function call is unspecified.

If *new_level* is zero, it causes the implementation to maintain the concurrency level at its discretion as if *pthread_setconcurrency*( ) had never been called.

The *pthread_getconcurrency*( ) function shall return the value set by a previous call to the *pthread_setconcurrency*( ) function. If the *pthread_setconcurrency*( ) function was not previously called, this function shall return zero to indicate that the implementation is maintaining the concurrency level.

A call to *pthread_setconcurrency*( ) shall inform the implementation of its desired concurrency level. The implementation shall use this as a hint, not a requirement.

If an implementation does not support multiplexing of user threads on top of several kernel-scheduled entities, the *pthread_setconcurrency*( ) and *pthread_getconcurrency*( ) functions are provided for source code compatibility but they shall have no effect when called. To maintain the function semantics, the *new_level* parameter is saved when *pthread_setconcurrency*( ) is called so that a subsequent call to *pthread_getconcurrency*( ) shall return the same value.

RETURN VALUE

If successful, the *pthread_setconcurrency*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread_getconcurrency*( ) function shall always return the concurrency level set by a previous call to *pthread_setconcurrency*( ). If the *pthread_setconcurrency*( ) function has never been called, *pthread_getconcurrency*( ) shall return zero.

ERRORS

The *pthread_setconcurrency*( ) function shall fail if:

[EINVAL] The value specified by *new_level* is negative.

[EAGAIN] The value specific by *new_level* would cause a system resource to be exceeded.

These functions shall not return an error code of [EINTR].

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

Use of these functions changes the state of the underlying concurrency upon which the application depends. Library developers are advised to not use the *pthread_getconcurrency*( ) and *pthread_setconcurrency*( ) functions since their use may conflict with an applications use of these functions.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME THREADS**)

SYNOPSIS

THR TCT
```
#include <pthread.h>
#include <time.h>

int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

DESCRIPTION

The *pthread_getcpuclockid*( ) function shall return in *clock_id* the clock ID of the CPU-time clock of the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

RETURN VALUE

Upon successful completion, *pthread_getcpuclockid*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_getcpuclockid*( ) function may fail if:

[ESRCH] The value specified by *thread_id* does not refer to an existing thread.

EXAMPLES

None.

APPLICATION USAGE

The *pthread_getcpuclockid*( ) function is part of the Thread CPU-Time Clocks option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getcpuclockid( ), *clock_getres*( ), *timer_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_getschedparam, pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

THR TPS
```
#include <pthread.h>

int pthread_getschedparam(pthread_t thread, int *restrict policy,
    struct sched_param *restrict param);
int pthread_setschedparam(pthread_t thread, int policy,
    const struct sched_param *param);
```

DESCRIPTION

The *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions shall, respectively, get and set the scheduling policy and parameters of individual threads within a multi-threaded process to be retrieved and set. For SCHED_FIFO and SCHED_RR, the only required member of the **sched_param** structure is the priority *sched_priority*. For SCHED_OTHER, the affected scheduling parameters are implementation-defined.

The *pthread_getschedparam*( ) function shall retrieve the scheduling policy and scheduling parameters for the thread whose thread ID is given by *thread* and shall store those values in *policy* and *param*, respectively. The priority value returned from *pthread_getschedparam*( ) shall be the value specified by the most recent *pthread_setschedparam*( ), *pthread_setschedprio*( ), or *pthread_create*( ) call affecting the target thread. It shall not reflect any temporary adjustments to its priority as a result of any priority inheritance or ceiling functions. The *pthread_setschedparam*( ) function shall set the scheduling policy and associated scheduling parameters for the thread whose thread ID is given by *thread* to the policy and associated parameters provided in *policy* and *param*, respectively.

The *policy* parameter may have the value SCHED_OTHER, SCHED_FIFO, or SCHED_RR. The scheduling parameters for the SCHED_OTHER policy are implementation-defined. The SCHED_FIFO and SCHED_RR policies shall have a single scheduling parameter, *priority*.

TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, then the *policy* argument may have the value SCHED_SPORADIC, with the exception for the *pthread_setschedparam*( ) function that if the scheduling policy was not SCHED_SPORADIC at the time of the call, it is implementation-defined whether the function is supported; in other words, the implementation need not allow the application to dynamically change the scheduling policy to SCHED_SPORADIC. The sporadic server scheduling policy has the associated parameters *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, *sched_priority*, and *sched_ss_max_repl*. The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail.

If the *pthread_setschedparam*( ) function fails, the scheduling parameters shall not be changed for the target thread.

RETURN VALUE

If successful, the *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33377 **ERRORS**

33378 The *pthread_getschedparam*( ) function may fail if:

33379 [ESRCH] The value specified by *thread* does not refer to an existing thread.

33380 The *pthread_setschedparam*( ) function may fail if:

33381 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated
33382 with the scheduling policy *policy* is invalid.

33383 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an
33384 unsupported value.

33385 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to
33386 SCHED_SPORADIC, and the implementation does not support this change.

33387 [EPERM] The caller does not have the appropriate permission to set either the
33388 scheduling parameters or the scheduling policy of the specified thread.

33389 [EPERM] The implementation does not allow the application to modify one of the
33390 parameters to the value specified.

33391 [ESRCH] The value specified by *thread* does not refer to a existing thread.

33392 These functions shall not return an error code of [EINTR].

33393 **EXAMPLES**

33394 None.

33395 **APPLICATION USAGE**

33396 None.

33397 **RATIONALE**

33398 None.

33399 **FUTURE DIRECTIONS**

33400 None.

33401 **SEE ALSO**

33402 *pthread_setschedprio*( ), *sched_getparam*( ), *sched_getscheduler*( ), the Base Definitions volume of
33403 IEEE Std 1003.1-2001, **<pthread.h>**, **<sched.h>**

33404 **CHANGE HISTORY**

33405 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33406 **Issue 6**

33407 The *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions are marked as part of the
33408 Threads and Thread Execution Scheduling options.

33409 The [ENOSYS] error condition has been removed as stubs need not be provided if an
33410 implementation does not support the Thread Execution Scheduling option.

33411 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the
33412 *pthread_setschedparam*( ) function so that its second argument is of type **int**.

33413 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

33414 The **restrict** keyword is added to the *pthread_getschedparam*( ) prototype for alignment with the
33415 ISO/IEC 9899: 1999 standard.

33416 The Open Group Corrigendum U047/1 is applied.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a call to the *pthread_setschedprio*( ) function.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_getspecific, pthread_setspecific — thread-specific data management

SYNOPSIS

THR

```
#include <pthread.h>

void *pthread_getspecific(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *value);
```

DESCRIPTION

The *pthread_getspecific*( ) function shall return the value currently bound to the specified *key* on behalf of the calling thread.

The *pthread_setspecific*( ) function shall associate a thread-specific *value* with a *key* obtained via a previous call to *pthread_key_create*( ). Different threads may bind different values to the same key. These values are typically pointers to blocks of dynamically allocated memory that have been reserved for use by the calling thread.

The effect of calling *pthread_getspecific*( ) or *pthread_setspecific*( ) with a *key* value not obtained from *pthread_key_create*( ) or after *key* has been deleted with *pthread_key_delete*( ) is undefined.

Both *pthread_getspecific*( ) and *pthread_setspecific*( ) may be called from a thread-specific data destructor function. A call to *pthread_getspecific*( ) for the thread-specific data key being destroyed shall return the value NULL, unless the value is changed (after the destructor starts) by a call to *pthread_setspecific*( ). Calling *pthread_setspecific*( ) from a thread-specific data destructor routine may result either in lost storage (after at least PTHREAD_DESTRUCTOR_ITERATIONS attempts at destruction) or in an infinite loop.

Both functions may be implemented as macros.

RETURN VALUE

The *pthread_getspecific*( ) function shall return the thread-specific data value associated with the given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be returned.

If successful, the *pthread_setspecific*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

No errors are returned from *pthread_getspecific*( ).

The *pthread_setspecific*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to associate the value with the key.

The *pthread_setspecific*( ) function may fail if:

[EINVAL] The key value is invalid.

These functions shall not return an error code of [EINTR].

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33455 **EXAMPLES**

33456 None.

33457 **APPLICATION USAGE**

33458 None.

33459 **RATIONALE**

33460 Performance and ease-of-use of *pthread_getspecific*( ) are critical for functions that rely on
33461 maintaining state in thread-specific data. Since no errors are required to be detected by it, and
33462 since the only error that could be detected is the use of an invalid key, the function to
33463 *pthread_getspecific*( ) has been designed to favor speed and simplicity over error reporting.

33464 **FUTURE DIRECTIONS**

33465 None.

33466 **SEE ALSO**

33467 *pthread_key_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

33468 **CHANGE HISTORY**

33469 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33470 **Issue 6**

33471 The *pthread_getspecific*( ) and *pthread_setspecific*( ) functions are marked as part of the Threads
33472 option.

33473 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_join — wait for thread termination

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **value_ptr);
```

DESCRIPTION

The *pthread_join*( ) function shall suspend execution of the calling thread until the target *thread* terminates, unless the target *thread* has already terminated. On return from a successful *pthread_join*( ) call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit*( ) by the terminating thread shall be made available in the location referenced by *value_ptr*. When a *pthread_join*( ) returns successfully, the target thread has been terminated. The results of multiple simultaneous calls to *pthread_join*( ) specifying the same target thread are undefined. If the thread calling *pthread_join*( ) is canceled, then the target thread shall not be detached.

It is unspecified whether a thread that has exited but remains unjoined counts against {PTHREAD_THREADS_MAX}.

RETURN VALUE

If successful, the *pthread_join*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_join*( ) function shall fail if:

[EINVAL] The implementation has detected that the value specified by *thread* does not refer to a joinable thread.

[ESRCH] No thread could be found corresponding to that specified by the given thread ID.

The *pthread_join*( ) function may fail if:

[EDEADLK] A deadlock was detected or the value of *thread* specifies the calling thread.

The *pthread_join*( ) function shall not return an error code of [EINTR].

EXAMPLES

An example of thread creation and deletion follows:

```
typedef struct {
    int *ar;
    long n;
} subarray;

void *
incer(void *arg)
{
    long i;

    for (i = 0; i < ((subarray *)arg)->n; i++)
        ((subarray *)arg)->ar[i]++;
}

int main(void)
{
    int        ar[1000000];
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        pthread_t  th1, th2;
        subarray   sb1, sb2;

        sb1.ar = &ar[0];
        sb1.n  = 500000;
        (void) pthread_create(&th1, NULL, incer, &sb1);

        sb2.ar = &ar[500000];
        sb2.n  = 500000;
        (void) pthread_create(&th2, NULL, incer, &sb2);

        (void) pthread_join(th1, NULL);
        (void) pthread_join(th2, NULL);
        return 0;
    }
```

APPLICATION USAGE

None.

RATIONALE

The *pthread_join*() function is a convenience that has proven useful in multi-threaded applications. It is true that a programmer could simulate this function if it were not provided by passing extra state as part of the argument to the *start_routine*(). The terminating thread would set a flag to indicate termination and broadcast a condition that is part of that state; a joining thread would wait on that condition variable. While such a technique would allow a thread to wait on more complex conditions (for example, waiting for multiple threads to terminate), waiting on individual thread termination is considered widely useful. Also, including the *pthread_join*() function in no way precludes a programmer from coding such complex waits. Thus, while not a primitive, including *pthread_join*() in this volume of IEEE Std 1003.1-2001 was considered valuable.

The *pthread_join*() function provides a simple mechanism allowing an application to wait for a thread to terminate. After the thread terminates, the application may then choose to clean up resources that were used by the thread. For instance, after *pthread_join*() returns, any application-provided stack storage could be reclaimed.

The *pthread_join*() or *pthread_detach*() function should eventually be called for every thread that is created with the *detachstate* attribute set to PTHREAD_CREATE_JOINABLE so that storage associated with the thread may be reclaimed.

The interaction between *pthread_join*() and cancellation is well-defined for the following reasons:

- The *pthread_join*() function, like all other non-async-cancel-safe functions, can only be called with deferred cancelability type.
- Cancellation cannot occur in the disabled cancelability state.

Thus, only the default cancelability state need be considered. As specified, either the *pthread_join*() call is canceled, or it succeeds, but not both. The difference is obvious to the application, since either a cancellation handler is run or *pthread_join*() returns. There are no race conditions since *pthread_join*() was called in the deferred cancelability state.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33561 **SEE ALSO**

33562 *pthread_create*( ), *wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

33563 **CHANGE HISTORY**

33564 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33565 **Issue 6**

33566 The *pthread_join*( ) function is marked as part of the Threads option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

pthread_key_create — thread-specific data key creation

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));
```

DESCRIPTION

The *pthread_key_create*( ) function shall create a thread-specific data key visible to all threads in the process. Key values provided by *pthread_key_create*( ) are opaque objects used to locate thread-specific data. Although the same key value may be used by different threads, the values bound to the key by *pthread_setspecific*( ) are maintained on a per-thread basis and persist for the life of the calling thread.

Upon key creation, the value NULL shall be associated with the new key in all active threads. Upon thread creation, the value NULL shall be associated with all defined keys in the new thread.

An optional destructor function may be associated with each key value. At thread exit, if a key value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with that key, the value of the key is set to NULL, and then the function pointed to is called with the previously associated value as its sole argument. The order of destructor calls is unspecified if more than one destructor exists for a thread when it exits.

If, after all the destructors have been called for all non-NULL values with associated destructors, there are still some non-NULL values with associated destructors, then the process is repeated. If, after at least {PTHREAD_DESTRUCTOR_ITERATIONS} iterations of destructor calls for outstanding non-NULL values, there are still some non-NULL values with associated destructors, implementations may stop calling destructors, or they may continue calling destructors until no non-NULL values with associated destructors exist, even though this might result in an infinite loop.

RETURN VALUE

If successful, the *pthread_key_create*( ) function shall store the newly created key value at **key* and shall return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_key_create*( ) function shall fail if:

[EAGAIN] The system lacked the necessary resources to create another thread-specific data key, or the system-imposed limit on the total number of keys per process {PTHREAD_KEYS_MAX} has been exceeded.

[ENOMEM] Insufficient memory exists to create the key.

The *pthread_key_create*( ) function shall not return an error code of [EINTR].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

The following example demonstrates a function that initializes a thread-specific data key when it is first called, and associates a thread-specific object with each calling thread, initializing this object when necessary.

```
static pthread_key_t key;
static pthread_once_t key_once = PTHREAD_ONCE_INIT;

static void
make_key()
{
    (void) pthread_key_create(&key, NULL);
}

func()
{
    void *ptr;

    (void) pthread_once(&key_once, make_key);
    if ((ptr = pthread_getspecific(key)) == NULL) {
        ptr = malloc(OBJECT_SIZE);
        ...
        (void) pthread_setspecific(key, ptr);
    }
    ...
}
```

Note that the key has to be initialized before *pthread_getspecific*( ) or *pthread_setspecific*( ) can be used. The *pthread_key_create*( ) call could either be explicitly made in a module initialization routine, or it can be done implicitly by the first call to a module as in this example. Any attempt to use the key before it is initialized is a programming error, making the code below incorrect.

```
static pthread_key_t key;

func()
{
    void *ptr;

   /* KEY NOT INITIALIZED!!!  THIS WON'T WORK!!! */
    if ((ptr = pthread_getspecific(key)) == NULL &&
        pthread_setspecific(key, NULL) != 0) {
        pthread_key_create(&key, NULL);
        ...
    }
}
```

APPLICATION USAGE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

RATIONALE

Destructor Functions

Normally, the value bound to a key on behalf of a particular thread is a pointer to storage allocated dynamically on behalf of the calling thread. The destructor functions specified with *pthread_key_create*() are intended to be used to free this storage when the thread exits. Thread cancellation cleanup handlers cannot be used for this purpose because thread-specific data may persist outside the lexical scope in which the cancellation cleanup handlers operate.

If the value associated with a key needs to be updated during the lifetime of the thread, it may be necessary to release the storage associated with the old value before the new value is bound. Although the *pthread_setspecific*() function could do this automatically, this feature is not needed often enough to justify the added complexity. Instead, the programmer is responsible for freeing the stale storage:

```
pthread_getspecific(key, &old);
new = allocate();
destructor(old);
pthread_setspecific(key, new);
```

Note: The above example could leak storage if run with asynchronous cancellation enabled. No such problems occur in the default cancellation state if no cancellation points occur between the get and set.

There is no notion of a destructor-safe function. If an application does not call *pthread_exit*() from a signal handler, or if it blocks any signal whose handler may call *pthread_exit*() while calling async-unsafe functions, all functions may be safely called from destructors.

Non-Idempotent Data Key Creation

There were requests to make *pthread_key_create*() idempotent with respect to a given *key* address parameter. This would allow applications to call *pthread_key_create*() multiple times for a given *key* address and be guaranteed that only one key would be created. Doing so would require the key value to be previously initialized (possibly at compile time) to a known null value and would require that implicit mutual-exclusion be performed based on the address and contents of the *key* parameter in order to guarantee that exactly one key would be created.

Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread_key_create*(). On many implementations, implicit mutual-exclusion would also have to be performed by *pthread_getspecific*() and *pthread_setspecific*() in order to guard against using incompletely stored or not-yet-visible key values. This could significantly increase the cost of important operations, particularly *pthread_getspecific*().

Thus, this proposal was rejected. The *pthread_key_create*() function performs no implicit synchronization. It is the responsibility of the programmer to ensure that it is called exactly once per key before use of the key. Several straightforward mechanisms can already be used to accomplish this, including calling explicit module initialization functions, using mutexes, and using *pthread_once*(). This places no significant burden on the programmer, introduces no possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows commonly used thread-specific data operations to be more efficient.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33686 **SEE ALSO**

33687 *pthread_getspecific*( ), *pthread_key_delete*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
33688 **<pthread.h>**

33689 **CHANGE HISTORY**

33690 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33691 **Issue 6**

33692 The *pthread_key_create*( ) function is marked as part of the Threads option.

33693 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_key_delete — thread-specific data key deletion

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_key_delete(pthread_key_t key);
```

DESCRIPTION

The *pthread_key_delete*( ) function shall delete a thread-specific data key previously returned by *pthread_key_create*( ). The thread-specific data values associated with *key* need not be NULL at the time *pthread_key_delete*( ) is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after *pthread_key_delete*( ) is called. Any attempt to use *key* following the call to *pthread_key_delete*( ) results in undefined behavior.

The *pthread_key_delete*( ) function shall be callable from within destructor functions. No destructor functions shall be invoked by *pthread_key_delete*( ). Any destructor function that may have been associated with *key* shall no longer be called upon thread exit.

RETURN VALUE

If successful, the *pthread_key_delete*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_key_delete*( ) function may fail if:

[EINVAL] The *key* value is invalid.

The *pthread_key_delete*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A thread-specific data key deletion function has been included in order to allow the resources associated with an unused thread-specific data key to be freed. Unused thread-specific data keys can arise, among other scenarios, when a dynamically loaded module that allocated a key is unloaded.

Conforming applications are responsible for performing any cleanup actions needed for data structures associated with the key to be deleted, including data referenced by thread-specific data values. No such cleanup is done by *pthread_key_delete*( ). In particular, destructor functions are not called. There are several reasons for this division of responsibility:

1. The associated destructor functions used to free thread-specific data at thread exit time are only guaranteed to work correctly when called in the thread that allocated the thread-specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot be used to free thread-specific data in other threads at key deletion time. Attempting to have them called by other threads at key deletion time would require other threads to be asynchronously interrupted. But since interrupted threads could be in an arbitrary state, including holding locks necessary for the destructor to run, this approach would fail. In general, there is no safe mechanism whereby an implementation could free thread-specific data at key deletion time.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2. Even if there were a means of safely freeing thread-specific data associated with keys to be deleted, doing so would require that implementations be able to enumerate the threads with non-NULL data and potentially keep them from creating more thread-specific data while the key deletion is occurring. This special case could cause extra synchronization in the normal case, which would otherwise be unnecessary.

For an application to know that it is safe to delete a key, it has to know that all the threads that might potentially ever use the key do not attempt to use it again. For example, it could know this if all the client threads have called a cleanup procedure declaring that they are through with the module that is being shut down, perhaps by setting a reference count to zero.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_create( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_key_delete*( ) function is marked as part of the Threads option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

pthread_kill — send a signal to a thread

SYNOPSIS

THR
```
#include <signal.h>

int pthread_kill(pthread_t thread, int sig);
```

DESCRIPTION

The *pthread_kill*( ) function shall request that a signal be delivered to the specified thread.

As in *kill*( ), if *sig* is zero, error checking shall be performed but no signal shall actually be sent.

RETURN VALUE

Upon successful completion, the function shall return a value of zero. Otherwise, the function shall return an error number. If the *pthread_kill*( ) function fails, no signal shall be sent.

ERRORS

The *pthread_kill*( ) function shall fail if:

[ESRCH] No thread could be found corresponding to that specified by the given thread ID.

[EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.

The *pthread_kill*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

The *pthread_kill*( ) function provides a mechanism for asynchronously directing a signal at a thread in the calling process. This could be used, for example, by one thread to affect broadcast delivery of a signal to a set of threads.

Note that *pthread_kill*( ) only causes the signal to be handled in the context of the given thread; the signal action (termination or stopping) affects the process as a whole.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

kill( ), *pthread_self*( ), *raise*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_kill*( ) function is marked as part of the Threads option.

The APPLICATION USAGE section is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_mutex_destroy, pthread_mutex_init — destroy and initialize a mutex

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

DESCRIPTION

The *pthread_mutex_destroy*( ) function shall destroy the mutex object referenced by *mutex*; the mutex object becomes, in effect, uninitialized. An implementation may cause *pthread_mutex_destroy*( ) to set the object referenced by *mutex* to an invalid value. A destroyed mutex object can be reinitialized using *pthread_mutex_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked mutex results in undefined behavior.

The *pthread_mutex_init*( ) function shall initialize the mutex referenced by *mutex* with attributes specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the same as passing the address of a default mutex attributes object. Upon successful initialization, the state of the mutex becomes initialized and unlocked.

Only *mutex* itself may be used for performing synchronization. The result of referring to copies of *mutex* in calls to *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), *pthread_mutex_unlock*( ), and *pthread_mutex_destroy*( ) is undefined.

Attempting to initialize an already initialized mutex results in undefined behavior.

In cases where default mutex attributes are appropriate, the macro PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes that are statically allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread_mutex_init*( ) with parameter *attr* specified as NULL, except that no error checks are performed.

RETURN VALUE

If successful, the *pthread_mutex_destroy*( ) and *pthread_mutex_init*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed immediately at the beginning of processing for the function and shall cause an error return prior to modifying the state of the mutex specified by *mutex*.

ERRORS

The *pthread_mutex_destroy*( ) function may fail if:

[EBUSY] The implementation has detected an attempt to destroy the object referenced by *mutex* while it is locked or referenced (for example, while being used in a *pthread_cond_timedwait*( ) or *pthread_cond_wait*( )) by another thread.

[EINVAL] The value specified by *mutex* is invalid.

The *pthread_mutex_init*( ) function shall fail if:

[EAGAIN] The system lacked the necessary resources (other than memory) to initialize another mutex.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[ENOMEM] Insufficient memory exists to initialize the mutex.

[EPERM] The caller does not have the privilege to perform the operation.

The *pthread_mutex_init*( ) function may fail if:

[EBUSY] The implementation has detected an attempt to reinitialize the object referenced by *mutex*, a previously initialized, but not yet destroyed, mutex.

[EINVAL] The value specified by *attr* is invalid.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Alternate Implementations Possible

This volume of IEEE Std 1003.1-2001 supports several alternative implementations of mutexes. An implementation may store the lock directly in the object of type **pthread_mutex_t**. Alternatively, an implementation may store the lock in the heap and merely store a pointer, handle, or unique ID in the mutex object. Either implementation has advantages or may be required on certain hardware configurations. So that portable code can be written that is invariant to this choice, this volume of IEEE Std 1003.1-2001 does not define assignment or equality for this type, and it uses the term ''initialize'' to reinforce the (more restrictive) notion that the lock may actually reside in the mutex object itself.

Note that this precludes an over-specification of the type of the mutex or condition variable and motivates the opaqueness of the type.

An implementation is permitted, but not required, to have *pthread_mutex_destroy*( ) store an illegal value into the mutex. This may help detect erroneous programs that try to lock (or otherwise reference) a mutex that has already been destroyed.

Tradeoff Between Error Checks and Performance Supported

Many of the error checks were made optional in order to let implementations trade off performance *versus* degree of error checking according to the needs of their specific applications and execution environment. As a general rule, errors or conditions caused by the system (such as insufficient memory) always need to be reported, but errors due to an erroneously coded application (such as failing to provide adequate synchronization to prevent a mutex from being deleted while in use) are made optional.

A wide range of implementations is thus made possible. For example, an implementation intended for application debugging may implement all of the error checks, but an implementation running a single, provably correct application under very tight performance constraints in an embedded computer might implement minimal checks. An implementation might even be provided in two versions, similar to the options that compilers provide: a full-checking, but slower version; and a limited-checking, but faster version. To forbid this optionality would be a disservice to users.

By carefully limiting the use of ''undefined behavior'' only to things that an erroneous (badly coded) application might do, and by defining that resource-not-available errors are mandatory, this volume of IEEE Std 1003.1-2001 ensures that a fully-conforming application is portable

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

across the full range of implementations, while not forcing all implementations to add overhead to check for numerous things that a correct program never does.

Why No Limits are Defined

Defining symbols for the maximum number of mutexes and condition variables was considered but rejected because the number of these objects may change dynamically. Furthermore, many implementations place these objects into application memory; thus, there is no explicit maximum.

Static Initializers for Mutexes and Condition Variables

Providing for static initialization of statically allocated synchronization objects allows modules with private static synchronization variables to avoid runtime initialization tests and overhead. Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in C libraries, where for various reasons the design calls for self-initialization instead of requiring an explicit module initialization function to be called. An example use of static initialization follows.

Without static initialization, a self-initializing routine *foo*( ) might look as follows:

```
static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
static pthread_mutex_t foo_mutex;

void foo_init()
{
    pthread_mutex_init(&foo_mutex, NULL);
}

void foo()
{
    pthread_once(&foo_once, foo_init);
    pthread_mutex_lock(&foo_mutex);
   /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

With static initialization, the same routine could be coded as follows:

```
static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;

void foo()
{
    pthread_mutex_lock(&foo_mutex);
   /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

Note that the static initialization both eliminates the need for the initialization test inside *pthread_once*( ) and the fetch of &*foo_mutex* to learn the address to be passed to *pthread_mutex_lock*( ) or *pthread_mutex_unlock*( ).

Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a large class of systems; those where the (entire) synchronization object can be stored in application memory.

Yet the locking performance question is likely to be raised for machines that require mutexes to be allocated out of special memory. Such machines actually have to have mutexes and possibly

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

condition variables contain pointers to the actual hardware locks. For static initialization to work on such machines, *pthread_mutex_lock*( ) also has to test whether or not the pointer to the actual lock has been allocated. If it has not, *pthread_mutex_lock*( ) has to initialize it before use. The reservation of such resources can be made when the program is loaded, and hence return codes have not been added to mutex locking and condition variable waiting to indicate failure to complete initialization.

This runtime test in *pthread_mutex_lock*( ) would at first seem to be extra work; an extra test is required to see whether the pointer has been initialized. On most machines this would actually be implemented as a fetch of the pointer, testing the pointer against zero, and then using the pointer if it has already been initialized. While the test might seem to add extra work, the extra effort of testing a register is usually negligible since no extra memory references are actually done. As more and more machines provide caches, the real expenses are memory references, not instructions executed.

Alternatively, depending on the machine architecture, there are often ways to eliminate *all* overhead in the most important case: on the lock operations that occur *after* the lock has been initialized. This can be done by shifting more overhead to the less frequent operation: initialization. Since out-of-line mutex allocation also means that an address has to be dereferenced to find the actual lock, one technique that is widely applicable is to have static initialization store a bogus value for that address; in particular, an address that causes a machine fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity checks can be done, and then the correct address for the actual lock can be filled in. Subsequent lock operations incur no extra overhead since they do not ‘‘fault’’. This is merely one technique that can be used to support static initialization, while not adversely affecting the performance of lock acquisition. No doubt there are other techniques that are highly machine-dependent.

The locking overhead for machines doing out-of-line mutex allocation is thus similar for modules being implicitly initialized, where it is improved for those doing mutex allocation entirely inline. The inline case is thus made much faster, and the out-of-line case is not significantly worse.

Besides the issue of locking performance for such machines, a concern is raised that it is possible that threads would serialize contending for initialization locks when attempting to finish initializing statically allocated mutexes. (Such finishing would typically involve taking an internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing the internal lock.) First, many implementations would reduce such serialization by hashing on the mutex address. Second, such serialization can only occur a bounded number of times. In particular, it can happen at most as many times as there are statically allocated synchronization objects. Dynamically allocated objects would still be initialized via *pthread_mutex_init*( ) or *pthread_cond_init*( ).

Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient performance for an application on some implementation, the application can avoid static initialization altogether by explicitly initializing all synchronization objects with the corresponding *pthread_*_init*( ) functions, which are supported by all implementations. An implementation can also document the tradeoffs and advise which initialization technique is more efficient for that particular implementation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Destroying Mutexes

A mutex can be destroyed immediately after it is unlocked. For example, consider the following code:

```
struct obj {
pthread_mutex_t om;
    int refcnt;
    ...
};

obj_done(struct obj *op)
{
    pthread_mutex_lock(&op->om);
    if (--op->refcnt == 0) {
        pthread_mutex_unlock(&op->om);
(A)     pthread_mutex_destroy(&op->om);
(B)     free(op);
    } else
(C)     pthread_mutex_unlock(&op->om);
}
```

In this case *obj* is reference counted and *obj_done*( ) is called whenever a reference to the object is dropped. Implementations are required to allow an object to be destroyed and freed and potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line C).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_getprioceiling( ), *pthread_mutex_lock*( ), *pthread_mutex_timedlock*( ), *pthread_mutexattr_getpshared*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutex_destroy*( ) and *pthread_mutex_init*( ) functions are marked as part of the Threads option.

The *pthread_mutex_timedlock*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

The **restrict** keyword is added to the *pthread_mutex_init*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_mutex_getprioceiling, pthread_mutex_setprioceiling — get and set the priority ceiling of a mutex (**REALTIME THREADS**)

SYNOPSIS

THR TPP

```
#include <pthread.h>

int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
    int *restrict prioceiling);
int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```

DESCRIPTION

The *pthread_mutex_getprioceiling*( ) function shall return the current priority ceiling of the mutex.

The *pthread_mutex_setprioceiling*( ) function shall either lock the mutex if it is unlocked, or block until it can successfully lock the mutex, then it shall change the mutex's priority ceiling and release the mutex. When the change is successful, the previous value of the priority ceiling shall be returned in *old_ceiling*. The process of locking the mutex need not adhere to the priority protect protocol.

If the *pthread_mutex_setprioceiling*( ) function fails, the mutex priority ceiling shall not be changed.

RETURN VALUE

If successful, the *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) functions may fail if:

[EINVAL] The priority requested by *prioceiling* is out of range.

[EINVAL] The value specified by *mutex* does not refer to a currently existing mutex.

[EPERM] The caller does not have the privilege to perform the operation.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_destroy( ), *pthread_mutex_lock*( ), *pthread_mutex_timedlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 6

The *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) functions are marked as part of the Threads and Thread Priority Protection options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection option.

The [ENOSYS] error denoting non-support of the priority ceiling protocol for mutexes has been removed. This is because if the implementation provides the functions (regardless of whether _POSIX_PTHREAD_PRIO_PROTECT is defined), they must function as in the DESCRIPTION and therefore the priority ceiling protocol for mutexes is supported.

The *pthread_mutex_timedlock*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) prototypes for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34061 **NAME**

34062 pthread_mutex_init — initialize a mutex

34063 **SYNOPSIS**

34064 THR
```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```
34068

34069 **DESCRIPTION**

34070 Refer to *pthread_mutex_destroy*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

The mutex object referenced by *mutex* shall be locked by calling *pthread_mutex_lock*( ). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

XSI If the mutex type is PTHREAD_MUTEX_NORMAL, deadlock detection shall not be provided. Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, undefined behavior results.

If the mutex type is PTHREAD_MUTEX_ERRORCHECK, then error checking shall be provided. If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error shall be returned.

If the mutex type is PTHREAD_MUTEX_RECURSIVE, then the mutex shall maintain the concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock count shall be set to one. Every time a thread relocks this mutex, the lock count shall be incremented by one. Each time the thread unlocks the mutex, the lock count shall be decremented by one. When the lock count reaches zero, the mutex shall become available for other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error shall be returned.

If the mutex type is PTHREAD_MUTEX_DEFAULT, attempting to recursively lock the mutex results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in undefined behavior.

The *pthread_mutex_trylock*( ) function shall be equivalent to *pthread_mutex_lock*( ), except that if the mutex object referenced by *mutex* is currently locked (by any thread, including the current thread), the call shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE and the mutex is currently owned by the calling thread, the mutex lock count shall be incremented by one and the *pthread_mutex_trylock*( ) function shall immediately return success.

XSI The *pthread_mutex_unlock*( ) function shall release the mutex object referenced by *mutex*. The manner in which a mutex is released is dependent upon the mutex's type attribute. If there are threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock*( ) is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

XSI (In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become available when the count reaches zero and the calling thread no longer has any locks on this mutex.)

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread shall resume waiting for the mutex as if it was not interrupted.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34117 **RETURN VALUE**

34118 If successful, the *pthread_mutex_lock*( ) and *pthread_mutex_unlock*( ) functions shall return zero;
34119 otherwise, an error number shall be returned to indicate the error.

34120 The *pthread_mutex_trylock*( ) function shall return zero if a lock on the mutex object referenced by
34121 *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

34122 **ERRORS**

34123 The *pthread_mutex_lock*( ) and *pthread_mutex_trylock*( ) functions shall fail if:

34124 [EINVAL] The *mutex* was created with the protocol attribute having the value
34125 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
34126 the mutex's current priority ceiling.

34127 The *pthread_mutex_trylock*( ) function shall fail if:

34128 [EBUSY] The *mutex* could not be acquired because it was already locked.

34129 The *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), and *pthread_mutex_unlock*( ) functions may
34130 fail if:

34131 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34132 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive
34133 locks for *mutex* has been exceeded.

34134 The *pthread_mutex_lock*( ) function may fail if:

34135 [EDEADLK] The current thread already owns the mutex.

34136 The *pthread_mutex_unlock*( ) function may fail if:

34137 [EPERM] The current thread does not own the mutex.

34138 These functions shall not return an error code of [EINTR].

34139 **EXAMPLES**

34140 None.

34141 **APPLICATION USAGE**

34142 None.

34143 **RATIONALE**

34144 Mutex objects are intended to serve as a low-level primitive from which other thread
34145 synchronization functions can be built. As such, the implementation of mutexes should be as
34146 efficient as possible, and this has ramifications on the features available at the interface.

34147 The mutex functions and the particular default settings of the mutex attributes have been
34148 motivated by the desire to not preclude fast, inlined implementations of mutex locking and
34149 unlocking.

34150 For example, deadlocking on a double-lock is explicitly allowed behavior in order to avoid
34151 requiring more overhead in the basic mechanism than is absolutely necessary. (More ''friendly''
34152 mutexes that detect deadlock or that allow multiple locking by the same thread are easily
34153 constructed by the user via the other mechanisms provided. For example, *pthread_self*( ) can be
34154 used to record mutex ownership.) Implementations might also choose to provide such extended
34155 features as options via special mutex attributes.

34156 Since most attributes only need to be checked when a thread is going to be blocked, the use of
34157 attributes does not slow the (common) mutex-locking case.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it would require storing the current thread ID when each mutex is locked, and this could incur unacceptable levels of overhead. Similar arguments apply to a *mutex_tryunlock* operation.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_destroy( ), *pthread_mutex_timedlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), and *pthread_mutex_unlock*( ) functions are marked as part of the Threads option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The behavior when attempting to relock a mutex is defined.

The *pthread_mutex_timedlock*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34176 **NAME**

34177 pthread_mutex_setprioceiling — change the priority ceiling of a mutex (**REALTIME**
34178 **THREADS**)

34179 **SYNOPSIS**

34180 THR TPP
```
#include <pthread.h>

int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```
34181
34182
34183

34184 **DESCRIPTION**

34185 Refer to *pthread_mutex_getprioceiling*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_mutex_timedlock — lock a mutex (**ADVANCED REALTIME**)

SYNOPSIS

THR TMO
```
#include <pthread.h>
#include <time.h>

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abs_timeout);
```

DESCRIPTION

The *pthread_mutex_timedlock*( ) function shall lock the mutex object referenced by *mutex*. If the mutex is already locked, the calling thread shall block until the mutex becomes available as in the *pthread_mutex_lock*( ) function. If the mutex cannot be locked without waiting for another thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout* has already been passed at the time of the call.

TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock; if the Timers option is not supported, the timeout shall be based on the system clock as returned by the *time*( ) function.

The resolution of the timeout shall be the resolution of the clock on which it is based. The **timespec** data type is defined in the **<time.h>** header.

Under no circumstance shall the function fail with a timeout if the mutex can be locked immediately. The validity of the *abs_timeout* parameter need not be checked if the mutex can be locked immediately.

As a consequence of the priority inheritance rules (for mutexes initialized with the PRIO_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this thread is no longer among the threads waiting for the mutex.

RETURN VALUE

If successful, the *pthread_mutex_timedlock*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_timedlock*( ) function shall fail if:

[EINVAL] The mutex was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex' current priority ceiling.

[EINVAL] The process or thread would have blocked, and the *abs_timeout* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

[ETIMEDOUT] The mutex could not be locked before the specified timeout expired.

The *pthread_mutex_timedlock*( ) function may fail if:

[EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34229 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive
34230 locks for *mutex* has been exceeded.

34231 [EDEADLK] The current thread already owns the mutex.

34232 This function shall not return an error code of [EINTR].

34233 **EXAMPLES**

34234 None.

34235 **APPLICATION USAGE**

34236 The *pthread_mutex_timedlock*( ) function is part of the Threads and Timeouts options and need
34237 not be provided on all implementations.

34238 **RATIONALE**

34239 None.

34240 **FUTURE DIRECTIONS**

34241 None.

34242 **SEE ALSO**

34243 *pthread_mutex_destroy*( ), *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), *time*( ), the Base
34244 Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<time.h>**

34245 **CHANGE HISTORY**

34246 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34247 **NAME**

34248 pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

34249 **SYNOPSIS**

34250 THR
```
#include <pthread.h>

int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```
34253

34254 **DESCRIPTION**

34255 Refer to *pthread_mutex_lock*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_mutexattr_destroy, pthread_mutexattr_init — destroy and initialize the mutex attributes object

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

DESCRIPTION

The *pthread_mutexattr_destroy*( ) function shall destroy a mutex attributes object; the object becomes, in effect, uninitialized. An implementation may cause *pthread_mutexattr_destroy*( ) to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be reinitialized using *pthread_mutexattr_init*( ); the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_mutexattr_init*( ) function shall initialize a mutex attributes object *attr* with the default value for all of the attributes defined by the implementation.

Results are undefined if *pthread_mutexattr_init*( ) is called specifying an already initialized *attr* attributes object.

After a mutex attributes object has been used to initialize one or more mutexes, any function affecting the attributes object (including destruction) shall not affect any previously initialized mutexes.

RETURN VALUE

Upon successful completion, *pthread_mutexattr_destroy*( ) and *pthread_mutexattr_init*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_destroy*( ) function may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *pthread_mutexattr_init*( ) function shall fail if:

[ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See *pthread_attr_init*( ) for a general explanation of attributes. Attributes objects allow implementations to experiment with useful extensions and permit extension of this volume of IEEE Std 1003.1-2001 without changing the existing functions. Thus, they provide for future extensibility of this volume of IEEE Std 1003.1-2001 and reduce the temptation to standardize prematurely on semantics that are not yet widely implemented or understood.

Examples of possible additional mutex attributes that have been discussed are *spin_only*, *limited_spin*, *no_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean: recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes would transparently keep records of queue length, wait time, and so on.) Since there is not yet

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

wide agreement on the usefulness of these resulting from shared implementation and usage experience, they are not yet specified in this volume of IEEE Std 1003.1-2001. Mutex attributes objects, however, make it possible to test out these concepts for possible standardization at a later time.

Mutex Attributes and Performance

Care has been taken to ensure that the default values of the mutex attributes have been defined such that mutexes initialized with the defaults have simple enough semantics so that the locking and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few other basic instructions).

There is at least one implementation method that can be used to reduce the cost of testing at lock-time if a mutex has non-default attributes. One such method that an implementation can employ (and this can be made fully transparent to fully conforming POSIX applications) is to secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to lock such a mutex causes the implementation to branch to the ''slow path'' as if the mutex were unavailable; then, on the slow path, the implementation can do the ''real work'' to lock a non-default mutex. The underlying unlock operation is more complicated since the implementation never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending on the hardware, there may be certain optimizations that can be used so that whatever mutex attributes are considered ''most frequently used'' can be processed most efficiently.

Process Shared Memory and Synchronization

The existence of memory mapping functions in this volume of IEEE Std 1003.1-2001 leads to the possibility that an application may allocate the synchronization objects from this section in memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

In order to permit such usage, while at the same time keeping the usual case (that is, usage within a single process) efficient, a *process-shared* option has been defined.

If an implementation supports the _POSIX_THREAD_PROCESS_SHARED option, then the *process-shared* attribute can be used to indicate that mutexes or condition variables may be accessed by threads of multiple processes.

The default setting of PTHREAD_PROCESS_PRIVATE has been chosen for the *process-shared* attribute so that the most efficient forms of these synchronization objects are created by default.

Synchronization variables that are initialized with the PTHREAD_PROCESS_PRIVATE *process-shared* attribute may only be operated on by threads in the process that initialized them. Synchronization variables that are initialized with the PTHREAD_PROCESS_SHARED *process-shared* attribute may be operated on by any thread in any process that has access to it. In particular, these processes may exist beyond the lifetime of the initializing process. For example, the following code implements a simple counting semaphore in a mapped file that may be used by many processes.

```
/* sem.h */
struct semaphore {
    pthread_mutex_t lock;
    pthread_cond_t nonzero;
    unsigned count;
};
typedef struct semaphore semaphore_t;

semaphore_t *semaphore_create(char *semaphore_name);
semaphore_t *semaphore_open(char *semaphore_name);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
34346       void semaphore_post(semaphore_t *semap);
34347       void semaphore_wait(semaphore_t *semap);
34348       void semaphore_close(semaphore_t *semap);

34349       /* sem.c */
34350       #include <sys/types.h>
34351       #include <sys/stat.h>
34352       #include <sys/mman.h>
34353       #include <fcntl.h>
34354       #include <pthread.h>
34355       #include "sem.h"

34356       semaphore_t *
34357       semaphore_create(char *semaphore_name)
34358       {
34359       int fd;
34360           semaphore_t *semap;
34361           pthread_mutexattr_t psharedm;
34362           pthread_condattr_t psharedc;

34363           fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
34364           if (fd < 0)
34365               return (NULL);
34366           (void) ftruncate(fd, sizeof(semaphore_t));
34367           (void) pthread_mutexattr_init(&psharedm);
34368           (void) pthread_mutexattr_setpshared(&psharedm,
34369               PTHREAD_PROCESS_SHARED);
34370           (void) pthread_condattr_init(&psharedc);
34371           (void) pthread_condattr_setpshared(&psharedc,
34372               PTHREAD_PROCESS_SHARED);
34373           semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34374                   PROT_READ | PROT_WRITE, MAP_SHARED,
34375                   fd, 0);
34376           close (fd);
34377           (void) pthread_mutex_init(&semap->lock, &psharedm);
34378           (void) pthread_cond_init(&semap->nonzero, &psharedc);
34379           semap->count = 0;
34380           return (semap);
34381       }

34382       semaphore_t *
34383       semaphore_open(char *semaphore_name)
34384       {
34385           int fd;
34386           semaphore_t *semap;

34387           fd = open(semaphore_name, O_RDWR, 0666);
34388           if (fd < 0)
34389               return (NULL);
34390           semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34391                   PROT_READ | PROT_WRITE, MAP_SHARED,
34392                   fd, 0);
34393           close (fd);
34394           return (semap);
34395       }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
34396       void
34397       semaphore_post(semaphore_t *semap)
34398       {
34399           pthread_mutex_lock(&semap->lock);
34400           if (semap->count == 0)
34401               pthread_cond_signal(&semapx->nonzero);
34402           semap->count++;
34403           pthread_mutex_unlock(&semap->lock);
34404       }

34405       void
34406       semaphore_wait(semaphore_t *semap)
34407       {
34408           pthread_mutex_lock(&semap->lock);
34409           while (semap->count == 0)
34410               pthread_cond_wait(&semap->nonzero, &semap->lock);
34411           semap->count--;
34412           pthread_mutex_unlock(&semap->lock);
34413       }

34414       void
34415       semaphore_close(semaphore_t *semap)
34416       {
34417           munmap((void *) semap, sizeof(semaphore_t));
34418       }
```

34419 The following code is for three separate processes that create, post, and wait on a semaphore in
34420 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and
34421 decrement the counting semaphore (waiting and waking as required) even though they did not
34422 initialize the semaphore.

```
34423       /* create.c */
34424       #include "pthread.h"
34425       #include "sem.h"

34426       int
34427       main()
34428       {
34429           semaphore_t *semap;

34430           semap = semaphore_create("/tmp/semaphore");
34431           if (semap == NULL)
34432               exit(1);
34433           semaphore_close(semap);
34434           return (0);
34435       }

34436       /* post */
34437       #include "pthread.h"
34438       #include "sem.h"

34439       int
34440       main()
34441       {
34442           semaphore_t *semap;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            semap = semaphore_open("/tmp/semaphore");
            if (semap == NULL)
                exit(1);
            semaphore_post(semap);
            semaphore_close(semap);
            return (0);
        }

        /* wait */
        #include "pthread.h"
        #include "sem.h"

        int
        main()
        {
            semaphore_t *semap;

            semap = semaphore_open("/tmp/semaphore");
            if (semap == NULL)
                exit(1);
            semaphore_wait(semap);
            semaphore_close(semap);
            return (0);
        }
```

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( ), *pthread_mutexattr_destroy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutexattr_destroy*( ) and *pthread_mutexattr_init*( ) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling — get and set the prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

THR TPP
```
#include <pthread.h>

int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *
    restrict attr, int *restrict prioceiling);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
```

DESCRIPTION

The *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions, respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*( ).

The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of *prioceiling* are within the maximum range of priorities defined by SCHED_FIFO.

The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum priority level at which the critical section guarded by the mutex is executed. In order to avoid priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are within the maximum range of priorities defined under the SCHED_FIFO scheduling policy.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions may fail if:

[EINVAL] The value specified by *attr* or *prioceiling* is invalid.

[EPERM] The caller does not have the privilege to perform the operation.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34516 **CHANGE HISTORY**

34517 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34518 Marked as part of the Realtime Threads Feature Group.

34519 **Issue 6**

34520 The *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions are
34521 marked as part of the Threads and Thread Priority Protection options.

34522 The [ENOSYS] error condition has been removed as stubs need not be provided if an
34523 implementation does not support the Thread Priority Protection option.

34524 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*
34525 argument.

34526 The **restrict** keyword is added to the *pthread_mutexattr_getprioceiling*( ) prototype for alignment
34527 with the ISO / IEC 9899: 1999 standard.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol — get and set the protocol attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

THR
```
#include <pthread.h>
```

TPP|TPI
```
int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *
    restrict attr, int *restrict protocol);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

DESCRIPTION

The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions, respectively, shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*( ).

The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of *protocol* may be one of:

PTHREAD_PRIO_NONE

TPI PTHREAD_PRIO_INHERIT

TPP PTHREAD_PRIO_PROTECT

which are defined in the **<pthread.h>** header.

When a thread owns a mutex with the PTHREAD_PRIO_NONE *protocol* attribute, its priority and scheduling shall not be affected by its mutex ownership.

TPI When a thread is blocking higher priority threads because of owning one or more mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of its priority or the priority of the highest priority thread waiting on any of the mutexes owned by this thread and initialized with this protocol.

TPP When a thread owns one or more mutexes initialized with the PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all the mutexes owned by this thread and initialized with this attribute, regardless of whether other threads are blocked on any of these mutexes or not.

While a thread is holding a mutex which has been initialized with the PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail of the scheduling queue at its priority in the event that its original priority is changed, such as by a call to *sched_setparam*( ). Likewise, when a thread unlocks a mutex that has been initialized with the PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail of the scheduling queue at its priority in the event that its original priority is changed.

If a thread simultaneously owns several mutexes initialized with different protocols, it shall execute at the highest of the priorities that it would have obtained by each of these protocols.

TPI When a thread makes a call to *pthread_mutex_lock*( ), the mutex was initialized with the protocol attribute having the value PTHREAD_PRIO_INHERIT, when the calling thread is blocked because the mutex is owned by another thread, that owner thread shall inherit the priority level of the calling thread as long as it continues to own the mutex. The implementation shall update its execution priority to the maximum of its assigned priority and all its inherited priorities. Furthermore, if this owner thread itself becomes blocked on another mutex, the same priority

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

inheritance effect shall be propagated to this other owner thread, in a recursive manner.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setprotocol*( ) function shall fail if:

[ENOTSUP] The value specified by *protocol* is an unsupported value.

The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions may fail if:

[EINVAL] The value specified by *attr* or *protocol* is invalid.

[EPERM] The caller does not have the privilege to perform the operation.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions are marked as part of the Threads option and either the Thread Priority Protection or Thread Priority Inheritance options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection or Thread Priority Inheritance options.

The **restrict** keyword is added to the *pthread_mutexattr_getprotocol*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_mutexattr_getpshared, pthread_mutexattr_setpshared — get and set the process-shared attribute

SYNOPSIS

THR TSH
```
#include <pthread.h>

int pthread_mutexattr_getpshared(const pthread_mutexattr_t *
    restrict attr, int *restrict pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_mutexattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*. The *pthread_mutexattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a mutex to be operated upon by any thread that has access to the memory where the mutex is allocated, even if the mutex is allocated in memory that is shared by multiple processes. If the *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the mutex shall only be operated upon by threads created within the same process as the thread that initialized the mutex; if threads of differing processes attempt to operate on such a mutex, the behavior is undefined. The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE.

RETURN VALUE

Upon successful completion, *pthread_mutexattr_setpshared*( ) shall return zero; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *pthread_mutexattr_getpshared*( ) shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_getpshared*( ) and *pthread_mutexattr_setpshared*( ) functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *pthread_mutexattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34651 **SEE ALSO**

34652 *pthread_cond_destroy*( ), *pthread_create*( ), *pthread_mutex_destroy*( ), *pthread_mutexattr_destroy*( ), the
34653 Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

34654 **CHANGE HISTORY**

34655 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34656 **Issue 6**

34657 The *pthread_mutexattr_getpshared*( ) and *pthread_mutexattr_setpshared*( ) functions are marked as
34658 part of the Threads and Thread Process-Shared Synchronization options.

34659 The **restrict** keyword is added to the *pthread_mutexattr_getpshared*( ) prototype for alignment
34660 with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

34661 **NAME**

34662 pthread_mutexattr_gettype, pthread_mutexattr_settype — get and set the mutex type attribute

34663 **SYNOPSIS**

34664 XSI
```
#include <pthread.h>

int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
    int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```
34668

34669 **DESCRIPTION**

34670 The *pthread_mutexattr_gettype*( ) and *pthread_mutexattr_settype*( ) functions, respectively, shall get
34671 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The
34672 default value of the *type* attribute is PTHREAD_MUTEX_DEFAULT.

34673 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types
34674 include:

34675 PTHREAD_MUTEX_NORMAL
34676 This type of mutex does not detect deadlock. A thread attempting to relock this mutex
34677 without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a
34678 different thread results in undefined behavior. Attempting to unlock an unlocked mutex
34679 results in undefined behavior.

34680 PTHREAD_MUTEX_ERRORCHECK
34681 This type of mutex provides error checking. A thread attempting to relock this mutex
34682 without first unlocking it shall return with an error. A thread attempting to unlock a mutex
34683 which another thread has locked shall return with an error. A thread attempting to unlock
34684 an unlocked mutex shall return with an error.

34685 PTHREAD_MUTEX_RECURSIVE
34686 A thread attempting to relock this mutex without first unlocking it shall succeed in locking
34687 the mutex. The relocking deadlock which can occur with mutexes of type
34688 PTHREAD_MUTEX_NORMAL cannot occur with this type of mutex. Multiple locks of this
34689 mutex shall require the same number of unlocks to release the mutex before another thread
34690 can acquire the mutex. A thread attempting to unlock a mutex which another thread has
34691 locked shall return with an error. A thread attempting to unlock an unlocked mutex shall
34692 return with an error.

34693 PTHREAD_MUTEX_DEFAULT
34694 Attempting to recursively lock a mutex of this type results in undefined behavior.
34695 Attempting to unlock a mutex of this type which was not locked by the calling thread
34696 results in undefined behavior. Attempting to unlock a mutex of this type which is not
34697 locked results in undefined behavior. An implementation may map this mutex to one of the
34698 other mutex types.

34699 **RETURN VALUE**

34700 Upon successful completion, the *pthread_mutexattr_gettype*( ) function shall return zero and store
34701 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,
34702 an error shall be returned to indicate the error.

34703 If successful, the *pthread_mutexattr_settype*( ) function shall return zero; otherwise, an error
34704 number shall be returned to indicate the error.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

34705 **ERRORS**

34706 The *pthread_mutexattr_settype*( ) function shall fail if:

34707 [EINVAL] The value *type* is invalid.

34708 The *pthread_mutexattr_gettype*( ) and *pthread_mutexattr_settype*( ) functions may fail if:

34709 [EINVAL] The value specified by *attr* is invalid.

34710 These functions shall not return an error code of [EINTR].

34711 **EXAMPLES**

34712 None.

34713 **APPLICATION USAGE**

34714 It is advised that an application should not use a PTHREAD_MUTEX_RECURSIVE mutex with
34715 condition variables because the implicit unlock performed for a *pthread_cond_timedwait*( ) or
34716 *pthread_cond_wait*( ) may not actually release the mutex (if it had been locked multiple times). If
34717 this happens, no other thread can satisfy the condition of the predicate.

34718 **RATIONALE**

34719 None.

34720 **FUTURE DIRECTIONS**

34721 None.

34722 **SEE ALSO**

34723 *pthread_cond_timedwait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

34724 **CHANGE HISTORY**

34725 First released in Issue 5.

34726 **Issue 6**

34727 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for
34728 *pthread_mutexattr_gettype*( ) is updated so that the first argument is of type **const**
34729 **pthread_mutexattr_t *.

34730 The **restrict** keyword is added to the *pthread_mutexattr_gettype*( ) prototype for alignment with
34731 the ISO/IEC 9899:1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

34732 **NAME**

34733 pthread_mutexattr_init — initialize the mutex attributes object

34734 **SYNOPSIS**

34735 THR
```
#include <pthread.h>
```

34736
```
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

34737

34738 **DESCRIPTION**

34739 Refer to *pthread_mutexattr_destroy*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34740 **NAME**

34741 pthread_mutexattr_setprioceiling — set the prioceiling attribute of the mutex attributes object
34742 (**REALTIME THREADS**)

34743 **SYNOPSIS**

34744 THR TPP
```
#include <pthread.h>

int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
```
34747

34748 **DESCRIPTION**

34749 Refer to *pthread_mutexattr_getprioceiling*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34750 **NAME**

34751 pthread_mutexattr_setprotocol — set the protocol attribute of the mutex attributes object
34752 (**REALTIME THREADS**)

34753 **SYNOPSIS**

34754 THR
```
#include <pthread.h>
```

34755 TPP|TPI
```
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

34756

34757

34758 **DESCRIPTION**

34759 Refer to *pthread_mutexattr_getprotocol*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34760 **NAME**

34761 pthread_mutexattr_setpshared — set the process-shared attribute

34762 **SYNOPSIS**

34763 THR TSH
```
#include <pthread.h>

int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```
34764

34765

34766

34767 **DESCRIPTION**

34768 Refer to *pthread_mutexattr_getpshared*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

34769 **NAME**

34770 pthread_mutexattr_settype — set the mutex type attribute

34771 **SYNOPSIS**

34772 XSI
```
#include <pthread.h>
```

34773
```
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

34774

34775 **DESCRIPTION**

34776 Refer to *pthread_mutexattr_gettype*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34777 **NAME**

34778 pthread_once — dynamic package initialization

34779 **SYNOPSIS**

34780 THR
```
#include <pthread.h>

int pthread_once(pthread_once_t *once_control,
    void (*init_routine)(void));
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```
34784

34785 **DESCRIPTION**

34786 The first call to *pthread_once*( ) by any thread in a process, with a given *once_control*, shall call the
34787 *init_routine* with no arguments. Subsequent calls of *pthread_once*( ) with the same *once_control*
34788 shall not call the *init_routine*. On return from *pthread_once*( ), *init_routine* shall have completed.
34789 The *once_control* parameter shall determine whether the associated initialization routine has
34790 been called.

34791 The *pthread_once*( ) function is not a cancellation point. However, if *init_routine* is a cancellation
34792 point and is canceled, the effect on *once_control* shall be as if *pthread_once*( ) was never called.

34793 The constant PTHREAD_ONCE_INIT is defined in the **<pthread.h>** header.

34794 The behavior of *pthread_once*( ) is undefined if *once_control* has automatic storage duration or is
34795 not initialized by PTHREAD_ONCE_INIT.

34796 **RETURN VALUE**

34797 Upon successful completion, *pthread_once*( ) shall return zero; otherwise, an error number shall
34798 be returned to indicate the error.

34799 **ERRORS**

34800 The *pthread_once*( ) function may fail if:

34801 [EINVAL] If either *once_control* or *init_routine* is invalid.

34802 The *pthread_once*( ) function shall not return an error code of [EINTR].

34803 **EXAMPLES**

34804 None.

34805 **APPLICATION USAGE**

34806 None.

34807 **RATIONALE**

34808 Some C libraries are designed for dynamic initialization. That is, the global initialization for the
34809 library is performed when the first procedure in the library is called. In a single-threaded
34810 program, this is normally implemented using a static variable whose value is checked on entry
34811 to a routine, as follows:

```
static int random_is_initialized = 0;
extern int initialize_random();

int random_function()
{
    if (random_is_initialized == 0) {
        initialize_random();
        random_is_initialized = 1;
    }
    ... /* Operations performed after initialization. */
}
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise, library initialization has to be accomplished by an explicit call to a library-exported initialization function prior to any use of the library.

For dynamic library initialization in a multi-threaded process, a simple initialization flag is not sufficient; the flag needs to be protected against modification by multiple threads simultaneously calling into the library. Protecting the flag requires the use of a mutex; however, mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized once requires a recursive solution to this problem.

The use of *pthread_once*( ) not only supplies an implementation-guaranteed means of dynamic initialization, it provides an aid to the reliable construction of multi-threaded and realtime systems. The preceding example then becomes:

```
#include <pthread.h>
static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
extern int initialize_random();

int random_function()
{
    (void) pthread_once(&random_is_initialized, initialize_random);
    ... /* Operations performed after initialization. */
}
```

Note that a **pthread_once_t** cannot be an array because some compilers do not accept the construct **&<array_name>**.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_once*( ) function is marked as part of the Threads option.

The [EINVAL] error is added as a may fail case for if either argument is invalid.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

34852 **NAME**

34853 pthread_rwlock_destroy, pthread_rwlock_init — destroy and initialize a read-write lock object

34854 **SYNOPSIS**

34855 THR
```
#include <pthread.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
    const pthread_rwlockattr_t *restrict attr);
```
34859

34860 **DESCRIPTION**

34861 The *pthread_rwlock_destroy*( ) function shall destroy the read-write lock object referenced by
34862 *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is
34863 undefined until the lock is reinitialized by another call to *pthread_rwlock_init*( ). An
34864 implementation may cause *pthread_rwlock_destroy*( ) to set the object referenced by *rwlock* to an
34865 invalid value. Results are undefined if *pthread_rwlock_destroy*( ) is called when any thread holds
34866 *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

34867 The *pthread_rwlock_init*( ) function shall allocate any resources required to use the read-write
34868 lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced
34869 by *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same
34870 as passing the address of a default read-write lock attributes object. Once initialized, the lock can
34871 be used any number of times without being reinitialized. Results are undefined if
34872 *pthread_rwlock_init*( ) is called specifying an already initialized read-write lock. Results are
34873 undefined if a read-write lock is used without first being initialized.

34874 If the *pthread_rwlock_init*( ) function fails, *rwlock* shall not be initialized and the contents of *rwlock*
34875 are undefined.

34876 Only the object referenced by *rwlock* may be used for performing synchronization. The result of
34877 referring to copies of that object in calls to *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ),
34878 *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_tryrdlock*( ),
34879 *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( ), or *pthread_rwlock_wrlock*( ) is undefined.

34880 **RETURN VALUE**

34881 If successful, the *pthread_rwlock_destroy*( ) and *pthread_rwlock_init*( ) functions shall return zero;
34882 otherwise, an error number shall be returned to indicate the error.

34883 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed
34884 immediately at the beginning of processing for the function and caused an error return prior to
34885 modifying the state of the read-write lock specified by *rwlock*.

34886 **ERRORS**

34887 The *pthread_rwlock_destroy*( ) function may fail if:

34888 [EBUSY] The implementation has detected an attempt to destroy the object referenced
34889 by *rwlock* while it is locked.

34890 [EINVAL] The value specified by *rwlock* is invalid.

34891 The *pthread_rwlock_init*( ) function shall fail if:

34892 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
34893 another read-write lock.

34894 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

34895 [EPERM] The caller does not have the privilege to perform the operation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *pthread_rwlock_init*( ) function may fail if:

[EBUSY] The implementation has detected an attempt to reinitialize the object referenced by *rwlock*, a previously initialized but not yet destroyed read-write lock.

[EINVAL] The value specified by *attr* is invalid.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these and related read-write lock functions may be subject to priority inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_rdlock( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_tryrdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( ), *pthread_rwlock_wrlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted from the SYNOPSIS.
- The DESCRIPTION is updated as follows:
  - — It explicitly notes allocation of resources upon initialization of a read-write lock object.
  - — A paragraph is added specifying that copies of read-write lock objects may not be used.
- An [EINVAL] error is added to the ERRORS section for *pthread_rwlock_init*( ), indicating that the *rwlock* value is invalid.
- The SEE ALSO section is updated.

The **restrict** keyword is added to the *pthread_rwlock_init*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION USAGE relating to priority inversion.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_rwlock_rdlock, pthread_rwlock_tryrdlock — lock a read-write lock object for reading

SYNOPSIS

THR

```
#include <pthread.h>

int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

DESCRIPTION

The *pthread_rwlock_rdlock*( ) function shall apply a read lock to the read-write lock referenced by *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are no writers blocked on the lock.

TPS

If the Thread Execution Scheduling option is supported, and the threads involved in the lock are executing with the scheduling policies SCHED_FIFO or SCHED_RR, the calling thread shall not acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock; otherwise, the calling thread shall acquire the lock.

TPS TSP

If the Threads Execution Scheduling option is supported, and the threads involved in the lock are executing with the SCHED_SPORADIC scheduling policy, the calling thread shall not acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock; otherwise, the calling thread shall acquire the lock.

If the Thread Execution Scheduling option is not supported, it is implementation-defined whether the calling thread acquires the lock when a writer does not hold the lock and there are writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the lock. The calling thread may deadlock if at the time the call is made it holds a write lock.

A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the *pthread_rwlock_rdlock*( ) function *n* times). If so, the application shall ensure that the thread performs matching unlocks (that is, it calls the *pthread_rwlock_unlock*( ) function *n* times).

The maximum number of simultaneous read locks that an implementation guarantees can be applied to a read-write lock shall be implementation-defined. The *pthread_rwlock_rdlock*( ) function may fail if this maximum would be exceeded.

The *pthread_rwlock_tryrdlock*( ) function shall apply a read lock as in the *pthread_rwlock_rdlock*( ) function, with the exception that the function shall fail if the equivalent *pthread_rwlock_rdlock*( ) call would have blocked the calling thread. In no case shall the *pthread_rwlock_tryrdlock*( ) function ever block; it always either acquires the lock or fails and returns immediately.

Results are undefined if any of these functions are called with an uninitialized read-write lock.

If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the signal handler the thread resumes waiting for the read-write lock for reading as if it was not interrupted.

RETURN VALUE

If successful, the *pthread_rwlock_rdlock*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread_rwlock_tryrdlock*( ) function shall return zero if the lock for reading on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

34978 **ERRORS**

34979 The *pthread_rwlock_tryrdlock*( ) function shall fail if:

34980 [EBUSY] The read-write lock could not be acquired for reading because a writer holds
34981 the lock or a writer with the appropriate priority was blocked on it.

34982 The *pthread_rwlock_rdlock*( ) and *pthread_rwlock_tryrdlock*( ) functions may fail if:

34983 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
34984 object.

34985 [EAGAIN] The read lock could not be acquired because the maximum number of read
34986 locks for *rwlock* has been exceeded.

34987 The *pthread_rwlock_rdlock*( ) function may fail if:

34988 [EDEADLK] The current thread already owns the read-write lock for writing.

34989 These functions shall not return an error code of [EINTR].

34990 **EXAMPLES**

34991 None.

34992 **APPLICATION USAGE**

34993 Applications using these functions may be subject to priority inversion, as discussed in the Base
34994 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

34995 **RATIONALE**

34996 None.

34997 **FUTURE DIRECTIONS**

34998 None.

34999 **SEE ALSO**

35000 *pthread_rwlock_destroy*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ),
35001 *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( ), *pthread_rwlock_wrlock*( ), the Base Definitions
35002 volume of IEEE Std 1003.1-2001, **<pthread.h>**

35003 **CHANGE HISTORY**

35004 First released in Issue 5.

35005 **Issue 6**

35006 The following changes are made for alignment with IEEE Std 1003.1j-2000:

35007 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
35008 now part of the Threads option (previously it was part of the Read-Write Locks option in
35009 IEEE Std 1003.1j-2000 and also part of the XSI extension).

35010 • The DESCRIPTION is updated as follows:

35011 — Conditions under which writers have precedence over readers are specified.

35012 — Failure of *pthread_rwlock_tryrdlock*( ) is clarified.

35013 — A paragraph on the maximum number of read locks is added.

35014 • In the ERRORS sections, [EBUSY] is modified to take into account write priority, and
35015 [EDEADLK] is deleted as a *pthread_rwlock_tryrdlock*( ) error.

35016 • The SEE ALSO section is updated.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35017 **NAME**

35018 pthread_rwlock_timedrdlock — lock a read-write lock for reading

35019 **SYNOPSIS**

35020 THR TMO
```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abs_timeout);
```
35024

35025 **DESCRIPTION**

35026 The *pthread_rwlock_timedrdlock*( ) function shall apply a read lock to the read-write lock
35027 referenced by *rwlock* as in the *pthread_rwlock_rdlock*( ) function. However, if the lock cannot be
35028 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
35029 when the specified timeout expires. The timeout shall expire when the absolute time specified
35030 by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the
35031 value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout*
35032 has already been passed at the time of the call.

35033 TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock. If
35034 the Timers option is not supported, the timeout shall be based on the system clock as returned
35035 by the *time*( ) function. The resolution of the timeout shall be the resolution of the clock on which
35036 it is based. The **timespec** data type is defined in the **<time.h>** header. Under no circumstances
35037 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the
35038 *abs_timeout* parameter need not be checked if the lock can be immediately acquired.

35039 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
35040 write lock via a call to *pthread_rwlock_timedrdlock*( ), upon return from the signal handler the
35041 thread shall resume waiting for the lock as if it was not interrupted.

35042 The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*.
35043 The results are undefined if this function is called with an uninitialized read-write lock.

35044 **RETURN VALUE**

35045 The *pthread_rwlock_timedrdlock*( ) function shall return zero if the lock for reading on the read-
35046 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned
35047 to indicate the error.

35048 **ERRORS**

35049 The *pthread_rwlock_timedrdlock*( ) function shall fail if:

35050 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35051 The *pthread_rwlock_timedrdlock*( ) function may fail if:

35052 [EAGAIN] The read lock could not be acquired because the maximum number of read
35053 locks for lock would be exceeded.

35054 [EDEADLK] The calling thread already holds a write lock on *rwlock*.

35055 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
35056 object, or the *abs_timeout* nanosecond value is less than zero or greater than or
35057 equal to 1 000 million.

35058 This function shall not return an error code of [EINTR].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35059 **EXAMPLES**

35060 None.

35061 **APPLICATION USAGE**

35062 Applications using this function may be subject to priority inversion, as discussed in the Base
35063 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35064 The *pthread_rwlock_timedrdlock*( ) function is part of the Threads and Timeouts options and need
35065 not be provided on all implementations.

35066 **RATIONALE**

35067 None.

35068 **FUTURE DIRECTIONS**

35069 None.

35070 **SEE ALSO**

35071 *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedwrlock*( ),
35072 *pthread_rwlock_tryrdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( ),
35073 *pthread_rwlock_wrlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**,
35074 **<time.h>**

35075 **CHANGE HISTORY**

35076 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35077 **NAME**

35078 pthread_rwlock_timedwrlock — lock a read-write lock for writing

35079 **SYNOPSIS**

35080 THR TMO
```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abs_timeout);
```
35084

35085 **DESCRIPTION**

35086 The *pthread_rwlock_timedwrlock*( ) function shall apply a write lock to the read-write lock
35087 referenced by *rwlock* as in the *pthread_rwlock_wrlock*( ) function. However, if the lock cannot be
35088 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
35089 when the specified timeout expires. The timeout shall expire when the absolute time specified
35090 by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the
35091 value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout*
35092 has already been passed at the time of the call.

35093 TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock. If
35094 the Timers option is not supported, the timeout shall be based on the system clock as returned
35095 by the *time*( ) function. The resolution of the timeout shall be the resolution of the clock on which
35096 it is based. The **timespec** data type is defined in the **<time.h>** header. Under no circumstances
35097 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the
35098 *abs_timeout* parameter need not be checked if the lock can be immediately acquired.

35099 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
35100 write lock via a call to *pthread_rwlock_timedwrlock*( ), upon return from the signal handler the
35101 thread shall resume waiting for the lock as if it was not interrupted.

35102 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The
35103 results are undefined if this function is called with an uninitialized read-write lock.

35104 **RETURN VALUE**

35105 The *pthread_rwlock_timedwrlock*( ) function shall return zero if the lock for writing on the read-
35106 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned
35107 to indicate the error.

35108 **ERRORS**

35109 The *pthread_rwlock_timedwrlock*( ) function shall fail if:

35110 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35111 The *pthread_rwlock_timedwrlock*( ) function may fail if:

35112 [EDEADLK] The calling thread already holds the *rwlock*.

35113 [EINVAL] The value specified by rwlock does not refer to an initialized read-write lock
35114 object, or the *abs_timeout* nanosecond value is less than zero or greater than or
35115 equal to 1 000 million.

35116 This function shall not return an error code of [EINTR].

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

The *pthread_rwlock_timedwrlock*( ) function is part of the Threads and Timeouts options and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_tryrdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( ), *pthread_rwlock_wrlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35135 **NAME**

35136 pthread_rwlock_tryrdlock — lock a read-write lock object for reading

35137 **SYNOPSIS**

35138 THR
```
#include <pthread.h>
```

35139
```
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

35140

35141 **DESCRIPTION**

35142 Refer to *pthread_rwlock_rdlock*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35143 **NAME**

35144 pthread_rwlock_trywrlock, pthread_rwlock_wrlock — lock a read-write lock object for writing

35145 **SYNOPSIS**

35146 THR
```
#include <pthread.h>

int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
```
35147

35148

35149

35150 **DESCRIPTION**

35151 The *pthread_rwlock_trywrlock*( ) function shall apply a write lock like the *pthread_rwlock_wrlock*( )
35152 function, with the exception that the function shall fail if any thread currently holds *rwlock* (for
35153 reading or writing).

35154 The *pthread_rwlock_wrlock*( ) function shall apply a write lock to the read-write lock referenced
35155 by *rwlock*. The calling thread acquires the write lock if no other thread (reader or writer) holds
35156 the read-write lock *rwlock*. Otherwise, the thread shall block until it can acquire the lock. The
35157 calling thread may deadlock if at the time the call is made it holds the read-write lock (whether a
35158 read or write lock).

35159 Implementations may favor writers over readers to avoid writer starvation.

35160 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35161 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the
35162 signal handler the thread resumes waiting for the read-write lock for writing as if it was not
35163 interrupted.

35164 **RETURN VALUE**

35165 The *pthread_rwlock_trywrlock*( ) function shall return zero if the lock for writing on the read-write
35166 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to
35167 indicate the error.

35168 If successful, the *pthread_rwlock_wrlock*( ) function shall return zero; otherwise, an error number
35169 shall be returned to indicate the error.

35170 **ERRORS**

35171 The *pthread_rwlock_trywrlock*( ) function shall fail if:

35172 [EBUSY] The read-write lock could not be acquired for writing because it was already
35173 locked for reading or writing.

35174 The *pthread_rwlock_trywrlock*( ) and *pthread_rwlock_wrlock*( ) functions may fail if:

35175 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
35176 object.

35177 The *pthread_rwlock_wrlock*( ) function may fail if:

35178 [EDEADLK] The current thread already owns the read-write lock for writing or reading.

35179 These functions shall not return an error code of [EINTR].

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35180 **EXAMPLES**

35181 None.

35182 **APPLICATION USAGE**

35183 Applications using these functions may be subject to priority inversion, as discussed in the Base
35184 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35185 **RATIONALE**

35186 None.

35187 **FUTURE DIRECTIONS**

35188 None.

35189 **SEE ALSO**

35190 *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ),
35191 *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_tryrdlock*( ), *pthread_rwlock_unlock*( ), the Base
35192 Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

35193 **CHANGE HISTORY**

35194 First released in Issue 5.

35195 **Issue 6**

35196 The following changes are made for alignment with IEEE Std 1003.1j-2000:

35197 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
35198 now part of the Threads option (previously it was part of the Read-Write Locks option in
35199 IEEE Std 1003.1j-2000 and also part of the XSI extension).

35200 • The [EDEADLK] error is deleted as a *pthread_rwlock_trywrlock*( ) error.

35201 • The SEE ALSO section is updated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_rwlock_unlock — unlock a read-write lock object

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

DESCRIPTION

The *pthread_rwlock_unlock*( ) function shall release a lock held on the read-write lock object referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the calling thread.

If this function is called to release a read lock from the read-write lock object and there are other read locks currently held on this read-write lock object, the read-write lock object remains in the read locked state. If this function releases the last read lock for this read-write lock object, the read-write lock object shall be put in the unlocked state with no owners.

If this function is called to release a write lock for this read-write lock object, the read-write lock object shall be put in the unlocked state.

If there are threads blocked on the lock when it becomes available, the scheduling policy shall TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is supported, when threads executing with the scheduling policies SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when the lock becomes available. For equal priority threads, write locks shall take precedence over read locks. If the Thread Execution Scheduling option is not supported, it is implementation-defined whether write locks take precedence over read locks.

Results are undefined if any of these functions are called with an uninitialized read-write lock.

RETURN VALUE

If successful, the *pthread_rwlock_unlock*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_unlock*( ) function may fail if:

[EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock object.

[EPERM] The current thread does not hold a lock on the read-write lock.

The *pthread_rwlock_unlock*( ) function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35243 **SEE ALSO**

35244 *pthread_rwlock_destroy*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ),
35245 *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_tryrdlock*( ), *pthread_rwlock_trywrlock*( ),
35246 *pthread_rwlock_wrlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

35247 **CHANGE HISTORY**

35248 First released in Issue 5.

35249 **Issue 6**

35250 The following changes are made for alignment with IEEE Std 1003.1j-2000:

35251 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
35252 now part of the Threads option (previously it was part of the Read-Write Locks option in
35253 IEEE Std 1003.1j-2000 and also part of the XSI extension).

35254 • The DESCRIPTION is updated as follows:

35255 — The conditions under which writers have precedence over readers are specified.

35256 — The concept of read-write lock owner is deleted.

35257 • The SEE ALSO section is updated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35258 **NAME**

35259 pthread_rwlock_wrlock — lock a read-write lock object for writing

35260 **SYNOPSIS**

35261 THR
```
#include <pthread.h>
```

35262
```
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
```

35263

35264 **DESCRIPTION**

35265 Refer to *pthread_rwlock_trywrlock*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35266 **NAME**

35267 pthread_rwlockattr_destroy, pthread_rwlockattr_init — destroy and initialize the read-write
35268 lock attributes object

35269 **SYNOPSIS**

35270 THR
```
#include <pthread.h>

int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```
35273

35274 **DESCRIPTION**

35275 The *pthread_rwlockattr_destroy*( ) function shall destroy a read-write lock attributes object. A
35276 destroyed *attr* attributes object can be reinitialized using *pthread_rwlockattr_init*( ); the results of
35277 otherwise referencing the object after it has been destroyed are undefined. An implementation
35278 may cause *pthread_rwlockattr_destroy*( ) to set the object referenced by *attr* to an invalid value.

35279 The *pthread_rwlockattr_init*( ) function shall initialize a read-write lock attributes object *attr* with
35280 the default value for all of the attributes defined by the implementation.

35281 Results are undefined if *pthread_rwlockattr_init*( ) is called specifying an already initialized *attr*
35282 attributes object.

35283 After a read-write lock attributes object has been used to initialize one or more read-write locks,
35284 any function affecting the attributes object (including destruction) shall not affect any previously
35285 initialized read-write locks.

35286 **RETURN VALUE**

35287 If successful, the *pthread_rwlockattr_destroy*( ) and *pthread_rwlockattr_init*( ) functions shall return
35288 zero; otherwise, an error number shall be returned to indicate the error.

35289 **ERRORS**

35290 The *pthread_rwlockattr_destroy*( ) function may fail if:

35291 [EINVAL] The value specified by *attr* is invalid.

35292 The *pthread_rwlockattr_init*( ) function shall fail if:

35293 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

35294 These functions shall not return an error code of [EINTR].

35295 **EXAMPLES**

35296 None.

35297 **APPLICATION USAGE**

35298 None.

35299 **RATIONALE**

35300 None.

35301 **FUTURE DIRECTIONS**

35302 None.

35303 **SEE ALSO**

35304 *pthread_rwlock_destroy*( ), *pthread_rwlockattr_getpshared*( ), *pthread_rwlockattr_setpshared*( ), the
35305 Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35306 **CHANGE HISTORY**

35307 First released in Issue 5.

35308 **Issue 6**

35309 The following changes are made for alignment with IEEE Std 1003.1j-2000:

35310 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
35311 now part of the Threads option (previously it was part of the Read-Write Locks option in
35312 IEEE Std 1003.1j-2000 and also part of the XSI extension).

35313 • The SEE ALSO section is updated.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared — get and set the process-shared attribute of the read-write lock attributes object

SYNOPSIS

THR TSH
```
#include <pthread.h>

int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *
    restrict attr, int *restrict pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_rwlockattr_getpshared*( ) function shall obtain the value of the *process-shared* attribute from the initialized attributes object referenced by *attr*. The *pthread_rwlockattr_setpshared*( ) function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute shall be set to PTHREAD_PROCESS_SHARED to permit a read-write lock to be operated upon by any thread that has access to the memory where the read-write lock is allocated, even if the read-write lock is allocated in memory that is shared by multiple processes. If the *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the read-write lock shall only be operated upon by threads created within the same process as the thread that initialized the read-write lock; if threads of differing processes attempt to operate on such a read-write lock, the behavior is undefined. The default value of the *process-shared* attribute shall be PTHREAD_PROCESS_PRIVATE.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

RETURN VALUE

Upon successful completion, the *pthread_rwlockattr_getpshared*( ) function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_rwlockattr_setpshared*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlockattr_getpshared*( ) and *pthread_rwlockattr_setpshared*( ) functions may fail if:

[EINVAL] The value specified by *attr* is invalid.

The *pthread_rwlockattr_setpshared*( ) function may fail if:

[EINVAL] The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of [EINTR].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlockattr_destroy*( ), *pthread_rwlockattr_init*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR TSH to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION notes that additional attributes are implementation-defined.
- The SEE ALSO section is updated.

The **restrict** keyword is added to the *pthread_rwlockattr_getpshared*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35373 **NAME**

35374 pthread_rwlockattr_init — initialize the read-write lock attributes object

35375 **SYNOPSIS**

35376 THR
```
#include <pthread.h>

int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```
35377

35378

35379 **DESCRIPTION**

35380 Refer to *pthread_rwlockattr_destroy*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35381 **NAME**

35382 pthread_rwlockattr_setpshared — set the process-shared attribute of the read-write lock
35383 attributes object

35384 **SYNOPSIS**

35385 THR TSH
```
#include <pthread.h>

int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```
35386
35387
35388

35389 **DESCRIPTION**

35390 Refer to *pthread_rwlockattr_getpshared*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35391 **NAME**

35392 pthread_self — get the calling thread ID

35393 **SYNOPSIS**

35394 THR
```
#include <pthread.h>
```

35395
```
pthread_t pthread_self(void);
```

35396

35397 **DESCRIPTION**

35398 The *pthread_self*( ) function shall return the thread ID of the calling thread.

35399 **RETURN VALUE**

35400 Refer to the DESCRIPTION.

35401 **ERRORS**

35402 No errors are defined.

35403 The *pthread_self*( ) function shall not return an error code of [EINTR].

35404 **EXAMPLES**

35405 None.

35406 **APPLICATION USAGE**

35407 None.

35408 **RATIONALE**

35409 The *pthread_self*( ) function provides a capability similar to the *getpid*( ) function for processes
35410 and the rationale is the same: the creation call does not provide the thread ID to the created
35411 thread.

35412 **FUTURE DIRECTIONS**

35413 None.

35414 **SEE ALSO**

35415 *pthread_create*( ), *pthread_equal*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
35416 **<pthread.h>**

35417 **CHANGE HISTORY**

35418 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

35419 **Issue 6**

35420 The *pthread_self*( ) function is marked as part of the Threads option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel — set cancelability state

SYNOPSIS

THR
```
#include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
void pthread_testcancel(void);
```

DESCRIPTION

The *pthread_setcancelstate*( ) function shall atomically both set the calling thread's cancelability state to the indicated *state* and return the previous cancelability state at the location referenced by *oldstate*. Legal values for *state* are PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DISABLE.

The *pthread_setcanceltype*( ) function shall atomically both set the calling thread's cancelability type to the indicated *type* and return the previous cancelability type at the location referenced by *oldtype*. Legal values for *type* are PTHREAD_CANCEL_DEFERRED and PTHREAD_CANCEL_ASYNCHRONOUS.

The cancelability state and type of any newly created threads, including the thread in which *main*( ) was first invoked, shall be PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED respectively.

The *pthread_testcancel*( ) function shall create a cancellation point in the calling thread. The *pthread_testcancel*( ) function shall have no effect if cancelability is disabled.

RETURN VALUE

If successful, the *pthread_setcancelstate*( ) and *pthread_setcanceltype*( ) functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setcancelstate*( ) function may fail if:

[EINVAL] The specified state is not PTHREAD_CANCEL_ENABLE or PTHREAD_CANCEL_DISABLE.

The *pthread_setcanceltype*( ) function may fail if:

[EINVAL] The specified type is not PTHREAD_CANCEL_DEFERRED or PTHREAD_CANCEL_ASYNCHRONOUS.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pthread_setcancelstate*( ) and *pthread_setcanceltype*( ) functions control the points at which a thread may be asynchronously canceled. For cancellation control to be usable in modular fashion, some rules need to be followed.

An object can be considered to be a generalization of a procedure. It is a set of procedures and global variables written as a unit and called by clients not known by the object. Objects may depend on other objects.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

First, cancelability should only be disabled on entry to an object, never explicitly enabled. On exit from an object, the cancelability state should always be restored to its value on entry to the object.

This follows from a modularity argument: if the client of an object (or the client of an object that uses that object) has disabled cancelability, it is because the client does not want to be concerned about cleaning up if the thread is canceled while executing some sequence of actions. If an object is called in such a state and it enables cancelability and a cancellation request is pending for that thread, then the thread is canceled, contrary to the wish of the client that disabled.

Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry to an object. But as with the cancelability state, on exit from an object the cancelability type should always be restored to its value on entry to the object.

Finally, only functions that are cancel-safe may be called from a thread that is asynchronously cancelable.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cancel( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_setcancelstate*( ), *pthread_setcanceltype*( ), and *pthread_testcancel*( ) functions are marked as part of the Threads option.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35487 **NAME**

35488 pthread_setconcurrency — set the level of concurrency

35489 **SYNOPSIS**

35490 XSI
```
#include <pthread.h>
```

35491
```
int pthread_setconcurrency(int new_level);
```

35492

35493 **DESCRIPTION**

35494 Refer to *pthread_getconcurrency*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35495 **NAME**

35496 pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME**
35497 **THREADS**)

35498 **SYNOPSIS**

35499 THR TPS
```
#include <pthread.h>

int pthread_setschedparam(pthread_t thread, int policy,
    const struct sched_param *param);
```
35500
35501
35502

35503 **DESCRIPTION**

35504 Refer to *pthread_getschedparam*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35505 **NAME**

35506 pthread_setschedprio — dynamic thread scheduling parameters access (**REALTIME**
35507 **THREADS**)

35508 **SYNOPSIS**

35509 THR TPS
```
#include <pthread.h>

int pthread_setschedprio(pthread_t thread, int prio);
```
35510
35511

35512 **DESCRIPTION**

35513 The *pthread_setschedprio*( ) function shall set the scheduling priority for the thread whose thread
35514 ID is given by *thread* to the value given by *prio*. See **Scheduling Policies** (on page 44) for a
35515 description on how this function call affects the ordering of the thread in the thread list for its
35516 new priority.

35517 If the *pthread_setschedprio*( ) function fails, the scheduling priority of the target thread shall not be
35518 changed.

35519 **RETURN VALUE**

35520 If successful, the *pthread_setschedprio*( ) function shall return zero; otherwise, an error number
35521 shall be returned to indicate the error.

35522 **ERRORS**

35523 The *pthread_setschedprio*( ) function may fail if:

35524 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.

35525 [ENOTSUP] An attempt was made to set the priority to an unsupported value.

35526 [EPERM] The caller does not have the appropriate permission to set the scheduling
35527 policy of the specified thread.

35528 [EPERM] The implementation does not allow the application to modify the priority to
35529 the value specified.

35530 [ESRCH] The value specified by *thread* does not refer to an existing thread.

35531 The *pthread_setschedprio*( ) function shall not return an error code of [EINTR].

35532 **EXAMPLES**

35533 None.

35534 **APPLICATION USAGE**

35535 None.

35536 **RATIONALE**

35537 The *pthread_setschedprio*( ) function provides a way for an application to temporarily raise its
35538 priority and then lower it again, without having the undesired side effect of yielding to other
35539 threads of the same priority. This is necessary if the application is to implement its own
35540 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This
35541 capability is especially important if the implementation does not support the Thread Priority
35542 Protection or Thread Priority Inheritance options, but even if those options are supported it is
35543 needed if the application is to bound priority inheritance for other resources, such as
35544 semaphores.

35545 The standard developers considered that while it might be preferable conceptually to solve this
35546 problem by modifying the specification of *pthread_setschedparam*( ), it was too late to make such a
35547 change, as there may be implementations that would need to be changed. Therefore, this new
35548 function was introduced.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35549 **FUTURE DIRECTIONS**

35550 None.

35551 **SEE ALSO**

35552 **Scheduling Policies** (on page 44), *pthread_getschedparam*( ), the Base Definitions volume of
35553 IEEE Std 1003.1-2001, **<pthread.h>**

35554 **CHANGE HISTORY**

35555 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35556 **NAME**

35557 pthread_setspecific — thread-specific data management

35558 **SYNOPSIS**

35559 THR
```
#include <pthread.h>
```

35560
```
int pthread_setspecific(pthread_key_t key, const void *value);
```

35561

35562 **DESCRIPTION**

35563 Refer to *pthread_getspecific*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

pthread_sigmask, sigprocmask — examine and change blocked signals

SYNOPSIS

```
#include <signal.h>

int pthread_sigmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
int sigprocmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
```

DESCRIPTION

The *pthread_sigmask*( ) function shall examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The function shall be equivalent to *sigprocmask*( ), without the restriction that the call be made in a single-threaded process.

In a single-threaded process, the *sigprocmask*( ) function shall examine or change (or both) the signal mask of the calling thread.

If the argument *set* is not a null pointer, it points to a set of signals to be used to change the currently blocked set.

The argument *how* indicates the way in which the set is changed, and the application shall ensure it consists of one of the following values:

SIG_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by *set*.

SIG_SETMASK The resulting set shall be the signal set pointed to by *set*.

SIG_UNBLOCK The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by *set*.

If the argument *oset* is not a null pointer, the previous mask shall be stored in the location pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the process' signal mask shall be unchanged; thus the call can be used to enquire about currently blocked signals.

If there are any pending unblocked signals after the call to *sigprocmask*( ), at least one of those signals shall be delivered before the call to *sigprocmask*( ) returns.

It is not possible to block those signals which cannot be ignored. This shall be enforced by the system without causing an error to be indicated.

If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked, the result is undefined, unless the signal was generated by the *kill*( ) function, the *sigqueue*( ) function, or the *raise*( ) function.

If *sigprocmask*( ) fails, the thread's signal mask shall not be changed.

The use of the *sigprocmask*( ) function is unspecified in a multi-threaded process.

RETURN VALUE

Upon successful completion *pthread_sigmask*( ) shall return 0; otherwise, it shall return the corresponding error number.

Upon successful completion, *sigprocmask*( ) shall return 0; otherwise, –1 shall be returned, *errno* shall be set to indicate the error, and the process' signal mask shall be unchanged.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35606 **ERRORS**

35607 THR The *pthread_sigmask*( ) and *sigprocmask*( ) functions shall fail if:

35608 [EINVAL] The value of the *how* argument is not equal to one of the defined values.

35609 THR The *pthread_sigmask*( ) function shall not return an error code of [EINTR].

35610 **EXAMPLES**

35611 None.

35612 **APPLICATION USAGE**

35613 None.

35614 **RATIONALE**

35615 When a process' signal mask is changed in a signal-catching function that is installed by
35616 *sigaction*( ), the restoration of the signal mask on return from the signal-catching function
35617 overrides that change (see *sigaction*( )). If the signal-catching function was installed with
35618 *signal*( ), it is unspecified whether this occurs.

35619 See *kill*( ) for a discussion of the requirement on delivery of signals.

35620 **FUTURE DIRECTIONS**

35621 None.

35622 **SEE ALSO**

35623 *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ),
35624 *sigqueue*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

35625 **CHANGE HISTORY**

35626 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

35627 **Issue 5**

35628 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

35629 The *pthread_sigmask*( ) function is added for alignment with the POSIX Threads Extension.

35630 **Issue 6**

35631 The *pthread_sigmask*( ) function is marked as part of the Threads option.

35632 The SYNOPSIS for *sigprocmask*( ) is marked as a CX extension to note that the presence of this
35633 function in the **<signal.h>** header is an extension to the ISO C standard.

35634 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

35635 • The DESCRIPTION is updated to explicitly state the functions which may generate the
35636 signal.

35637 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

35638 The **restrict** keyword is added to the *pthread_sigmask*( ) and *sigprocmask*( ) prototypes for
35639 alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35640 **NAME**

35641 pthread_spin_destroy, pthread_spin_init — destroy or initialize a spin lock object (**ADVANCED**
35642 **REALTIME THREADS**)

35643 **SYNOPSIS**

35644 THR SPI
```
#include <pthread.h>

int pthread_spin_destroy(pthread_spinlock_t *lock);
int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
```
35647

35648 **DESCRIPTION**

35649 The *pthread_spin_destroy*( ) function shall destroy the spin lock referenced by *lock* and release any
35650 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is
35651 reinitialized by another call to *pthread_spin_init*( ). The results are undefined if
35652 *pthread_spin_destroy*( ) is called when a thread holds the lock, or if this function is called with an
35653 uninitialized thread spin lock.

35654 The *pthread_spin_init*( ) function shall allocate any resources required to use the spin lock
35655 referenced by *lock* and initialize the lock to an unlocked state.

35656 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
35657 PTHREAD_PROCESS_SHARED, the implementation shall permit the spin lock to be operated
35658 upon by any thread that has access to the memory where the spin lock is allocated, even if it is
35659 allocated in memory that is shared by multiple processes.

35660 If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
35661 PTHREAD_PROCESS_PRIVATE, or if the option is not supported, the spin lock shall only be
35662 operated upon by threads created within the same process as the thread that initialized the spin
35663 lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is
35664 undefined.

35665 The results are undefined if *pthread_spin_init*( ) is called specifying an already initialized spin
35666 lock. The results are undefined if a spin lock is used without first being initialized.

35667 If the *pthread_spin_init*( ) function fails, the lock is not initialized and the contents of *lock* are
35668 undefined.

35669 Only the object referenced by *lock* may be used for performing synchronization.

35670 The result of referring to copies of that object in calls to *pthread_spin_destroy*( ),
35671 *pthread_spin_lock*( ), *pthread_spin_trylock*( ), or *pthread_spin_unlock*( ) is undefined.

35672 **RETURN VALUE**

35673 Upon successful completion, these functions shall return zero; otherwise, an error number shall
35674 be returned to indicate the error.

35675 **ERRORS**

35676 These functions may fail if:

35677 [EBUSY] The implementation has detected an attempt to initialize or destroy a spin
35678 lock while it is in use (for example, while being used in a *pthread_spin_lock*( )
35679 call) by another thread.

35680 [EINVAL] The value specified by *lock* is invalid.

35681 The *pthread_spin_init*( ) function shall fail if:

35682 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35683 [ENOMEM] Insufficient memory exists to initialize the lock.

35684 These functions shall not return an error code of [EINTR].

35685 **EXAMPLES**

35686 None.

35687 **APPLICATION USAGE**

35688 The *pthread_spin_destroy*( ) and *pthread_spin_init*( ) functions are part of the Spin Locks option
35689 and need not be provided on all implementations.

35690 **RATIONALE**

35691 None.

35692 **FUTURE DIRECTIONS**

35693 None.

35694 **SEE ALSO**

35695 *pthread_spin_lock*( ), *pthread_spin_unlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
35696 **<pthread.h>**

35697 **CHANGE HISTORY**

35698 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35699 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_spin_lock, pthread_spin_trylock — lock a spin lock object (**ADVANCED REALTIME THREADS**)

SYNOPSIS

THR SPI

```
#include <pthread.h>

int pthread_spin_lock(pthread_spinlock_t *lock);
int pthread_spin_trylock(pthread_spinlock_t *lock);
```

DESCRIPTION

The *pthread_spin_lock*( ) function shall lock the spin lock referenced by *lock*. The calling thread shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is, shall not return from the *pthread_spin_lock*( ) call) until the lock becomes available. The results are undefined if the calling thread holds the lock at the time the call is made. The *pthread_spin_trylock*( ) function shall lock the spin lock referenced by *lock* if it is not held by any thread. Otherwise, the function shall fail.

The results are undefined if any of these functions is called with an uninitialized spin lock.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value specified by *lock* does not refer to an initialized spin lock object.

The *pthread_spin_lock*( ) function may fail if:

[EDEADLK] The calling thread already holds the lock.

The *pthread_spin_trylock*( ) function shall fail if:

[EBUSY] A thread currently holds the lock.

These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

The *pthread_spin_lock*( ) and *pthread_spin_trylock*( ) functions are part of the Spin Locks option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_destroy( ), *pthread_spin_unlock*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35741 **CHANGE HISTORY**

35742 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35743 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35744 **NAME**

35745 pthread_spin_unlock — unlock a spin lock object (**ADVANCED REALTIME THREADS**)

35746 **SYNOPSIS**

35747 THR SPI
```
#include <pthread.h>

int pthread_spin_unlock(pthread_spinlock_t *lock);
```
35748

35749

35750 **DESCRIPTION**

35751 The *pthread_spin_unlock*( ) function shall release the spin lock referenced by *lock* which was
35752 locked via the *pthread_spin_lock*( ) or *pthread_spin_trylock*( ) functions. The results are undefined if
35753 the lock is not held by the calling thread. If there are threads spinning on the lock when
35754 *pthread_spin_unlock*( ) is called, the lock becomes available and an unspecified spinning thread
35755 shall acquire the lock.

35756 The results are undefined if this function is called with an uninitialized thread spin lock.

35757 **RETURN VALUE**

35758 Upon successful completion, the *pthread_spin_unlock*( ) function shall return zero; otherwise, an
35759 error number shall be returned to indicate the error.

35760 **ERRORS**

35761 The *pthread_spin_unlock*( ) function may fail if:

35762 [EINVAL] An invalid argument was specified.

35763 [EPERM] The calling thread does not hold the lock.

35764 This function shall not return an error code of [EINTR].

35765 **EXAMPLES**

35766 None.

35767 **APPLICATION USAGE**

35768 The *pthread_spin_unlock*( ) function is part of the Spin Locks option and need not be provided on
35769 all implementations.

35770 **RATIONALE**

35771 None.

35772 **FUTURE DIRECTIONS**

35773 None.

35774 **SEE ALSO**

35775 *pthread_spin_destroy*( ), *pthread_spin_lock*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
35776 **<pthread.h>**

35777 **CHANGE HISTORY**

35778 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35779 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pthread_testcancel — set cancelability state

SYNOPSIS

THR

```
#include <pthread.h>

void pthread_testcancel(void);
```

DESCRIPTION

Refer to *pthread_setcancelstate*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ptsname — get name of the slave pseudo-terminal device

SYNOPSIS

XSI
```
#include <stdlib.h>

char *ptsname(int fildes);
```

DESCRIPTION

The *ptsname*( ) function shall return the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the master device. The *ptsname*( ) function shall return a pointer to a string containing the pathname of the corresponding slave device.

The *ptsname*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *ptsname*( ) shall return a pointer to a string which is the name of the pseudo-terminal slave device. Upon failure, *ptsname*( ) shall return a null pointer. This could occur if *fildes* is an invalid file descriptor or if the slave device name does not exist in the file system.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The value returned may point to a static data area that is overwritten by each call to *ptsname*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

grantpt( ), *open*( ), *ttyname*( ), *unlockpt*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35824 **NAME**

35825 putc — put a byte on a stream

35826 **SYNOPSIS**

35827 `#include <stdio.h>`

35828 `int putc(int c, FILE *stream);`

35829 **DESCRIPTION**

35830 CX The functionality described on this reference page is aligned with the ISO C standard. Any
35831 conflict between the requirements described here and the ISO C standard is unintentional. This
35832 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

35833 The *putc*( ) function shall be equivalent to *fputc*( ), except that if it is implemented as a macro it
35834 may evaluate *stream* more than once, so the argument should never be an expression with side
35835 effects.

35836 **RETURN VALUE**

35837 Refer to *fputc*( ).

35838 **ERRORS**

35839 Refer to *fputc*( ).

35840 **EXAMPLES**

35841 None.

35842 **APPLICATION USAGE**

35843 Since it may be implemented as a macro, *putc*( ) may treat a *stream* argument with side effects
35844 incorrectly. In particular, *putc*(*c*,*f*++) does not necessarily work correctly. Therefore, use of this
35845 function is not recommended in such situations; *fputc*( ) should be used instead.

35846 **RATIONALE**

35847 None.

35848 **FUTURE DIRECTIONS**

35849 None.

35850 **SEE ALSO**

35851 *fputc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

35852 **CHANGE HISTORY**

35853 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35854 **NAME**

35855 putc_unlocked — stdio with explicit client locking

35856 **SYNOPSIS**

35857 TSF
```
#include <stdio.h>
```

35858
```
int putc_unlocked(int c, FILE *stream);
```

35859

35860 **DESCRIPTION**

35861 Refer to *getc_unlocked*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

35862 **NAME**

35863 putchar — put a byte on a stdout stream

35864 **SYNOPSIS**

35865 `#include <stdio.h>`

35866 `int putchar(int c);`

35867 **DESCRIPTION**

35868 CX The functionality described on this reference page is aligned with the ISO C standard. Any
35869 conflict between the requirements described here and the ISO C standard is unintentional. This
35870 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

35871 The function call *putchar*(*c*) shall be equivalent to *putc*(*c*,*stdout*).

35872 **RETURN VALUE**

35873 Refer to *fputc*( ).

35874 **ERRORS**

35875 Refer to *fputc*( ).

35876 **EXAMPLES**

35877 None.

35878 **APPLICATION USAGE**

35879 None.

35880 **RATIONALE**

35881 None.

35882 **FUTURE DIRECTIONS**

35883 None.

35884 **SEE ALSO**

35885 *putc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

35886 **CHANGE HISTORY**

35887 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35888 **NAME**

35889 putchar_unlocked — stdio with explicit client locking

35890 **SYNOPSIS**

35891 TSF
```
#include <stdio.h>
```

35892
```
int putchar_unlocked(int c);
```

35893

35894 **DESCRIPTION**

35895 Refer to *getc_unlocked*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35896 **NAME**

35897 putenv — change or add a value to an environment

35898 **SYNOPSIS**

35899 XSI
```
#include <stdlib.h>

int putenv(char *string);
```
35900

35901

35902 **DESCRIPTION**

35903 The *putenv*( ) function shall use the *string* argument to set environment variable values. The
35904 *string* argument should point to a string of the form "*name=value*". The *putenv*( ) function shall
35905 make the value of the environment variable *name* equal to *value* by altering an existing variable
35906 or creating a new one. In either case, the string pointed to by *string* shall become part of the
35907 environment, so altering the string shall change the environment. The space used by *string* is no
35908 longer used once a new string which defines *name* is passed to *putenv*( ).

35909 The *putenv*( ) function need not be reentrant. A function that is not required to be reentrant is not
35910 required to be thread-safe.

35911 **RETURN VALUE**

35912 Upon successful completion, *putenv*( ) shall return 0; otherwise, it shall return a non-zero value
35913 and set *errno* to indicate the error.

35914 **ERRORS**

35915 The *putenv*( ) function may fail if:

35916 [ENOMEM] Insufficient memory was available.

35917 **EXAMPLES**

35918 **Changing the Value of an Environment Variable**

35919 The following example changes the value of the *HOME* environment variable to the value
35920 **/usr/home**.

```
35921 #include <stdlib.h>
35922 ...
35923 static char *var = "HOME=/usr/home";
35924 int ret;

35925 ret = putenv(var);
```

35926 **APPLICATION USAGE**

35927 The *putenv*( ) function manipulates the environment pointed to by *environ*, and can be used in
35928 conjunction with *getenv*( ).

35929 See *exec*, for restrictions on changing the environment in multi-threaded applications.

35930 This routine may use *malloc*( ) to enlarge the environment.

35931 A potential error is to call *putenv*( ) with an automatic variable as the argument, then return from
35932 the calling function while *string* is still part of the environment.

35933 The *setenv*( ) function is preferred over this function.

35934 **RATIONALE**

35935 The standard developers noted that *putenv*( ) is the only function available to add to the
35936 environment without permitting memory leaks.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

35937 **FUTURE DIRECTIONS**

35938 None.

35939 **SEE ALSO**

35940 *exec*, *getenv*( ), *malloc*( ), *setenv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

35941 **CHANGE HISTORY**

35942 First released in Issue 1. Derived from Issue 1 of the SVID.

35943 **Issue 5**

35944 The type of the argument to this function is changed from **const char *** to **char ***. This was
35945 indicated as a FUTURE DIRECTION in previous issues.

35946 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35947 **Issue 6**

35948 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the
35949 DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to
35950 *exec*.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

35951 **NAME**

35952 putmsg, putpmsg — send a message on a STREAM (**STREAMS**)

35953 **SYNOPSIS**

35954 XSR
```
#include <stropts.h>

int putmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int flags);
int putpmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int band, int flags);
```
35959

35960 **DESCRIPTION**

35961 The *putmsg*( ) function shall create a message from a process buffer(s) and send the message to a
35962 STREAMS file. The message may contain either a data part, a control part, or both. The data and
35963 control parts are distinguished by placement in separate buffers, as described below. The
35964 semantics of each part are defined by the STREAMS module that receives the message.

35965 The *putpmsg*( ) function is equivalent to *putmsg*( ), except that the process can send messages in
35966 different priority bands. Except where noted, all requirements on *putmsg*( ) also pertain to
35967 *putpmsg*( ).

35968 The *fildes* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and
35969 *dataptr* arguments each point to a **strbuf** structure.

35970 The *ctlptr* argument points to the structure describing the control part, if any, to be included in
35971 the message. The *buf* member in the **strbuf** structure points to the buffer where the control
35972 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*
35973 member is not used by *putmsg*( ). In a similar manner, the argument *dataptr* specifies the data, if
35974 any, to be included in the message. The *flags* argument indicates what type of message should be
35975 sent and is described further below.

35976 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer
35977 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the
35978 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part
35979 shall be sent if either *dataptr*(*ctlptr*) is a null pointer or the *len* member of *dataptr*(*ctlptr*) is set to
35980 –1.

35981 For *putmsg*( ), if a control part is specified and *flags* is set to RS_HIPRI, a high priority message
35982 shall be sent. If no control part is specified, and *flags* is set to RS_HIPRI, *putmsg*( ) shall fail and
35983 set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be
35984 sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be
35985 sent and 0 shall be returned.

35986 For *putpmsg*( ), the flags are different. The *flags* argument is a bitmask with the following
35987 mutually-exclusive flags defined: MSG_HIPRI and MSG_BAND. If *flags* is set to 0, *putpmsg*( )
35988 shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG_HIPRI
35989 and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG_HIPRI and either
35990 no control part is specified or *band* is set to a non-zero value, *putpmsg*( ) shall fail and set *errno* to
35991 [EINVAL]. If *flags* is set to MSG_BAND, then a message shall be sent in the priority band
35992 specified by *band*. If a control part and data part are not specified and *flags* is set to MSG_BAND,
35993 no message shall be sent and 0 shall be returned.

35994 The *putmsg*( ) function shall block if the STREAM write queue is full due to internal flow control
35995 conditions, with the following exceptions:

35996 • For high-priority messages, *putmsg*( ) shall not block on this condition and continues
35997 processing the message.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

-·-',,·-·-'·,,·',·',,'---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- For other messages, *putmsg*( ) shall not block but shall fail when the write queue is full and O_NONBLOCK is set.

The *putmsg*( ) function shall also block, unless prevented by lack of internal resources, while waiting for the availability of message blocks in the STREAM, regardless of priority or whether O_NONBLOCK has been specified. No partial message shall be sent.

RETURN VALUE

Upon successful completion, *putmsg*( ) and *putpmsg*( ) shall return 0; otherwise, they shall return –1 and set *errno* to indicate the error.

ERRORS

The *putmsg*( ) and *putpmsg*( ) functions shall fail if:

[EAGAIN] A non-priority message was specified, the O_NONBLOCK flag is set, and the STREAM write queue is full due to internal flow control conditions; or buffers could not be allocated for the message that was to be created.

[EBADF] *fildes* is not a valid file descriptor open for writing.

[EINTR] A signal was caught during *putmsg*( ).

[EINVAL] An undefined value is specified in *flags*, or *flags* is set to RS_HIPRI or MSG_HIPRI and no control part is supplied, or the STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer, or *flags* is set to MSG_HIPRI and *band* is non-zero (for *putpmsg*( ) only).

[ENOSR] Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

[ENOSTR] A STREAM is not associated with *fildes*.

[ENXIO] A hangup condition was generated downstream for the specified STREAM.

[EPIPE] or [EIO] The *fildes* argument refers to a STREAMS-based pipe and the other end of the pipe is closed. A SIGPIPE signal is generated for the calling thread.

[ERANGE] The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message.

In addition, *putmsg*( ) and *putpmsg*( ) shall fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of *errno* does not reflect the result of *putmsg*( ) or *putpmsg*( ), but reflects the prior error.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Sending a High-Priority Message

The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg*( ) does the following:

1. Creates a high-priority message with a control part and a data part, using the buffers pointed to by *ctrlbuf* and *databuf*, respectively.
2. Sends the message to the STREAMS file identified by *fd*.

```
#include <stropts.h>
#include <string.h>
...
int fd;
char *ctrlbuf = "This is the control part";
char *databuf = "This is the data part";
struct strbuf ctrl;
struct strbuf data;
int ret;

ctrl.buf = ctrlbuf;
ctrl.len = strlen(ctrlbuf);

data.buf = databuf;
data.len = strlen(databuf);

ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

Using putpmsg( )

This example has the same effect as the previous example. In this example, however, the *putpmsg*( ) function creates and sends the message to the STREAMS file.

```
#include <stropts.h>
#include <string.h>
...
int fd;
char *ctrlbuf = "This is the control part";
char *databuf = "This is the data part";
struct strbuf ctrl;
struct strbuf data;
int ret;

ctrl.buf = ctrlbuf;
ctrl.len = strlen(ctrlbuf);

data.buf = databuf;
data.len = strlen(databuf);

ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

APPLICATION USAGE

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

36073 **RATIONALE**

36074 None.

36075 **FUTURE DIRECTIONS**

36076 None.

36077 **SEE ALSO**

36078 Section 2.6 (on page 38), *getmsg*( ), *poll*( ), *read*( ), *write*( ), the Base Definitions volume of
36079 IEEE Std 1003.1-2001, **<stropts.h>**

36080 **CHANGE HISTORY**

36081 First released in Issue 4, Version 2.

36082 **Issue 5**

36083 Moved from X/OPEN UNIX extension to BASE.

36084 The following text is removed from the DESCRIPTION: ‘‘The STREAM head guarantees that the
36085 control part of a message generated by *putmsg*( ) is at least 64 bytes in length’’.

36086 **Issue 6**

36087 This function is marked as part of the XSI STREAMS Option Group.

36088 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

puts — put a string on standard output

SYNOPSIS

```
#include <stdio.h>

int puts(const char *s);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *puts*( ) function shall write the string pointed to by *s*, followed by a <newline>, to the standard output stream *stdout*. The terminating null byte shall not be written.

CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful execution of *puts*( ) and the next successful completion of a call to *fflush*( ) or *fclose*( ) on the same stream or a call to *exit*( ) or *abort*( ).

RETURN VALUE

Upon successful completion, *puts*( ) shall return a non-negative number. Otherwise, it shall CX return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

ERRORS

Refer to *fputc*( ).

EXAMPLES

Printing to Standard Output

The following example gets the current time, converts it to a string using *localtime*( ) and *asctime*( ), and prints it to standard output using *puts*( ). It then prints the number of minutes to an event for which it is waiting.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",
    minutes_to_event);
...
```

APPLICATION USAGE

The *puts*( ) function appends a <newline>, while *fputs*( ) does not.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

36131 **SEE ALSO**

36132 *fopen*( ), *fputs*( ), *putc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

36133 **CHANGE HISTORY**

36134 First released in Issue 1. Derived from Issue 1 of the SVID.

36135 **Issue 6**

36136 Extensions beyond the ISO C standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pututxline — put an entry into the user accounting database

SYNOPSIS

XSI

```
#include <utmpx.h>

struct utmpx *pututxline(const struct utmpx *utmpx);
```

DESCRIPTION

Refer to *endutxent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

putwc — put a wide character on a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t putwc(wchar_t wc, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *putwc*( ) function shall be equivalent to *fputwc*( ), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side effects.

RETURN VALUE

Refer to *fputwc*( ).

ERRORS

Refer to *fputwc*( ).

EXAMPLES

None.

APPLICATION USAGE

Since it may be implemented as a macro, *putwc*( ) may treat a *stream* argument with side effects incorrectly. In particular, *putwc*(*wc*,*f*++) need not work correctly. Therefore, use of this function is not recommended; *fputwc*( ) should be used instead.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fputwc( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4.

Issue 5

Aligned with ISO/IEC 9899: 1990/Amendment 1: 1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

The Optional Header (OH) marking is removed from **<stdio.h>**.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

putwchar — put a wide character on a stdout stream

SYNOPSIS

```
#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The function call *putwchar*(*wc*) shall be equivalent to *putwc*(*wc*,*stdout*).

RETURN VALUE

Refer to *fputwc*( ).

ERRORS

Refer to *fputwc*( ).

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fputwc( ), *putwc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

36209 **NAME**

36210 pwrite — write on a file

36211 **SYNOPSIS**

36212 `#include <unistd.h>`

36213 XSI `ssize_t pwrite(int fildes, const void *buf, size_t nbyte,`

36214 `    off_t offset);`

36215

36216 **DESCRIPTION**

36217 Refer to *write*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

36218 **NAME**

36219 qsort — sort a table of data

36220 **SYNOPSIS**

36221 `#include <stdlib.h>`

36222 `void qsort(void *`*`base`*`, size_t `*`nel`*`, size_t `*`width`*`,`
36223 `    int (*`*`compar`*`)(const void *, const void *));`

36224 **DESCRIPTION**

36225 CX The functionality described on this reference page is aligned with the ISO C standard. Any
36226 conflict between the requirements described here and the ISO C standard is unintentional. This
36227 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36228 The *qsort*( ) function shall sort an array of *nel* objects, the initial element of which is pointed to by
36229 *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has
36230 the value zero, the comparison function pointed to by *compar* shall not be called and no
36231 rearrangement shall take place.

36232 The application shall ensure that the comparison function pointed to by *compar* does not alter the
36233 contents of the array. The implementation may reorder elements of the array between calls to the
36234 comparison function, but shall not alter the contents of any individual element.

36235 When the same objects (consisting of width bytes, irrespective of their current positions in the
36236 array) are passed more than once to the comparison function, the results shall be consistent with
36237 one another. That is, they shall define a total ordering on the array.

36238 The contents of the array shall be sorted in ascending order according to a comparison function.
36239 The *compar* argument is a pointer to the comparison function, which is called with two
36240 arguments that point to the elements being compared. The application shall ensure that the
36241 function returns an integer less than, equal to, or greater than 0, if the first argument is
36242 considered respectively less than, equal to, or greater than the second. If two members compare
36243 as equal, their order in the sorted array is unspecified.

36244 **RETURN VALUE**

36245 The *qsort*( ) function shall not return a value.

36246 **ERRORS**

36247 No errors are defined.

36248 **EXAMPLES**

36249 None.

36250 **APPLICATION USAGE**

36251 The comparison function need not compare every byte, so arbitrary data may be contained in
36252 the elements in addition to the values being compared.

36253 **RATIONALE**

36254 The requirement that each argument (hereafter referred to as *p)* to the comparison function is a
36255 pointer to elements of the array implies that for every call, for each argument separately, all of
36256 the following expressions are nonzero:

36257 `((char *)p - (char *)base) % width == 0`
36258 `(char *)p >= (char *)base`
36259 `(char *)p < (char *)base + nel * width`

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The RATIONALE is also updated. These changes are for alignment with the ISO C standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

36271 **NAME**

36272 raise — send a signal to the executing process

36273 **SYNOPSIS**

36274
```
#include <signal.h>
```

36275
```
int raise(int sig);
```

36276 **DESCRIPTION**

36277 CX The functionality described on this reference page is aligned with the ISO C standard. Any
36278 conflict between the requirements described here and the ISO C standard is unintentional. This
36279 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36280 CX The *raise*( ) function shall send the signal *sig* to the executing thread or process. If a signal
36281 handler is called, the *raise*( ) function shall not return until after the signal handler does.

36282 THR If the implementation supports the Threads option, the effect of the *raise*( ) function shall be
36283 equivalent to calling:

36284
```
pthread_kill(pthread_self(), sig);
```

36285

36286 CX Otherwise, the effect of the *raise*( ) function shall be equivalent to calling:

36287
```
kill(getpid(), sig);
```

36288

36289 **RETURN VALUE**

36290 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned
36291 and *errno* shall be set to indicate the error.

36292 **ERRORS**

36293 The *raise*( ) function shall fail if:

36294 CX [EINVAL] The value of the *sig* argument is an invalid signal number.

36295 **EXAMPLES**

36296 None.

36297 **APPLICATION USAGE**

36298 None.

36299 **RATIONALE**

36300 The term ‘‘thread’’ is an extension to the ISO C standard.

36301 **FUTURE DIRECTIONS**

36302 None.

36303 **SEE ALSO**

36304 *kill*( ), *sigaction*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**,
36305 **<sys/types.h>**

36306 **CHANGE HISTORY**

36307 First released in Issue 4. Derived from the ANSI C standard.

36308 **Issue 5**

36309 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EINVAL] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

rand, rand_r, srand — pseudo-random number generator

SYNOPSIS

```
#include <stdlib.h>

int rand(void);
int rand_r(unsigned *seed);
void srand(unsigned seed);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *rand*( ) function shall compute a sequence of pseudo-random integers in the range XSI [0,{RAND_MAX}] with a period of at least 2^{32}.

CX The *rand*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

TSF The *rand_r*( ) function shall compute a sequence of pseudo-random integers in the range [0,{RAND_MAX}]. (The value of the {RAND_MAX} macro shall be at least 32 767.)

If *rand_r*( ) is called with the same initial value for the object pointed to by *seed* and that object is not modified between successive returns and calls to *rand_r*( ), the same sequence shall be generated.

The *srand*( ) function uses the argument as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to *rand*( ). If *srand*( ) is then called with the same seed value, the sequence of pseudo-random numbers shall be repeated. If *rand*( ) is called before any calls to *srand*( ) are made, the same sequence shall be generated as when *srand*( ) is first called with a seed value of 1.

The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *rand*( ) or *srand*( ).

RETURN VALUE

The *rand*( ) function shall return the next pseudo-random number in the sequence.

TSF The *rand_r*( ) function shall return a pseudo-random integer.

The *srand*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

Generating a Pseudo-Random Number Sequence

The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
#include <stdio.h>
#include <stdlib.h>
...
    long count, i;
    char *keystr;
    int elementlen, len;
    char c;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
...
/* Initial random number generator. */
    srand(1);

    /* Create keys using only lowercase characters */
    len = 0;
    for (i=0; i<count; i++) {
        while (len < elementlen) {
            c = (char) (rand() % 128);
            if (islower(c))
                keystr[len++] = c;
        }

        keystr[len] = '\0';
        printf("%s Element%0*ld\n", keystr, elementlen, i);
        len = 0;
    }
```

Generating the Same Sequence on Different Machines

The following code defines a pair of functions that could be incorporated into applications wishing to ensure that the same sequence of numbers is generated across different machines.

```
static unsigned long next = 1;
int myrand(void)  /* RAND_MAX assumed to be 32767. */
{
    next = next * 1103515245 + 12345;
    return((unsigned)(next/65536) % 32768);
}

void mysrand(unsigned seed)
{
    next = seed;
}
```

APPLICATION USAGE

The *drand48*( ) function provides a much more elaborate random number generator.

The limitations on the amount of state that can be carried between one function call and another mean the *rand_r*( ) function can never be implemented in a way which satisfies all of the requirements on a pseudo-random number generator. Therefore this function should be avoided whenever non-trivial requirements (including safety) have to be fulfilled.

RATIONALE

The ISO C standard *rand*( ) and *srand*( ) functions allow per-process pseudo-random streams shared by all threads. Those two functions need not change, but there has to be mutual-exclusion that prevents interference between two threads concurrently accessing the random number generator.

With regard to *rand*( ), there are two different behaviors that may be wanted in a multi-threaded program:

1. A single per-process sequence of pseudo-random numbers that is shared by all threads that call *rand*( )
2. A different sequence of pseudo-random numbers for each thread that calls *rand*( )

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

This is provided by the modified thread-safe function based on whether the seed value is global to the entire process or local to each thread.

This does not address the known deficiencies of the *rand*( ) function implementations, which have been approached by maintaining more state. In effect, this specifies new thread-safe forms of a deficient function.

FUTURE DIRECTIONS

None.

SEE ALSO

drand48( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *rand_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *rand*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

Extensions beyond the ISO C standard are marked.

The *rand_r*( ) function is marked as part of the Thread-Safe Functions option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

random — generate pseudo-random number

SYNOPSIS

XSI

```
#include <stdlib.h>

long random(void);
```

DESCRIPTION

Refer to *initstate*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

36428 **NAME**

36429 pread, read — read from a file

36430 **SYNOPSIS**

36431 `#include <unistd.h>`

36432 XSI `ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);`

36433 `ssize_t read(int fildes, void *buf, size_t nbyte);`

36434 **DESCRIPTION**

36435 The *read*( ) function shall attempt to read *nbyte* bytes from the file associated with the open file
36436 descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on
36437 the same pipe, FIFO, or terminal device is unspecified.

36438 Before any action described below is taken, and if *nbyte* is zero, the *read*( ) function may detect
36439 and return errors as described below. In the absence of errors, or if error detection is not
36440 performed, the *read*( ) function shall return zero and have no other results.

36441 On files that support seeking (for example, a regular file), the *read*( ) shall start at a position in
36442 the file given by the file offset associated with *fildes*. The file offset shall be incremented by the
36443 number of bytes actually read.

36444 Files that do not support seeking—for example, terminals—always read from the current
36445 position. The value of a file offset associated with such a file is undefined.

36446 No data transfer shall occur past the current end-of-file. If the starting position is at or after the
36447 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent
36448 *read*( ) requests is implementation-defined.

36449 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

36450 When attempting to read from an empty pipe or FIFO:

36451 • If no process has the pipe open for writing, *read*( ) shall return 0 to indicate end-of-file.

36452 • If some process has the pipe open for writing and O_NONBLOCK is set, *read*( ) shall return
36453 –1 and set *errno* to [EAGAIN].

36454 • If some process has the pipe open for writing and O_NONBLOCK is clear, *read*( ) shall block
36455 the calling thread until some data is written or the pipe is closed by all processes that had the
36456 pipe open for writing.

36457 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and
36458 has no data currently available:

36459 • If O_NONBLOCK is set, *read*( ) shall return –1 and set *errno* to [EAGAIN].

36460 • If O_NONBLOCK is clear, *read*( ) shall block the calling thread until some data becomes
36461 available.

36462 • The use of the O_NONBLOCK flag has no effect if there is some data available.

36463 The *read*( ) function reads data previously written to a file. If any portion of a regular file prior to
36464 the end-of-file has not been written, *read*( ) shall return bytes with value 0. For example, *lseek*( )
36465 allows the file offset to be set beyond the end of existing data in the file. If data is later written at
36466 this point, subsequent reads in the gap between the previous end of data and the newly written
36467 data shall return bytes with value 0 until data is written into the gap.

36468 Upon successful completion, where *nbyte* is greater than 0, *read*( ) shall mark for update the
36469 *st_atime* field of the file, and shall return the number of bytes read. This number shall never be
36470 greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

file is less than *nbyte*, if the *read*( ) request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a *read*( ) from a file associated with a terminal may return one typed line of data.

If a *read*( ) is interrupted by a signal before it reads any data, it shall return –1 with *errno* set to [EINTR].

If a *read*( ) is interrupted by a signal after it has successfully read some data, it shall return the number of bytes read.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *fildes*.

If *fildes* refers to a socket, *read*( ) shall be equivalent to *recv*( ) with no flags set.

SIO If the O_DSYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If the O_SYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

SHM If *fildes* refers to a shared memory object, the result of the *read*( ) function is unspecified.

TYM If *fildes* refers to a typed memory object, the result of the *read*( ) function is unspecified.

XSR A *read*( ) from a STREAMS file can read data in three different modes: *byte-stream* mode, *message-nondiscard* mode, and *message-discard* mode. The default shall be byte-stream mode. This can be changed using the I_SRDOPT *ioctl*( ) request, and can be tested with I_GRDOPT *ioctl*( ). In byte-stream mode, *read*( ) shall retrieve data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, *read*( ) shall retrieve data until as many bytes as were requested are transferred, or until a message boundary is reached. If *read*( ) does not retrieve all the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by the next *read*( ) call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the *read*( ) returns shall be discarded, and shall not be available for a subsequent *read*( ), *getmsg*( ), or *getpmsg*( ) call.

How *read*( ) handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, *read*( ) shall accept data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The *read*( ) function shall then return the number of bytes read, and place the zero-byte message back on the STREAM to be retrieved by the next *read*( ), *getmsg*( ), or *getpmsg*( ). In message-nondiscard mode or message-discard mode, a zero-byte message shall return 0 and the message shall be removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message shall be removed from the STREAM and 0 shall be returned, regardless of the read mode.

A *read*( ) from a STREAMS file shall return the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMs are in control-normal mode, in which a *read*( ) from a STREAMS file can only process messages that contain a data part but do not contain a control part. The *read*( ) shall fail if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the I_SRDOPT *ioctl*( ) command. In control-data mode, *read*( ) shall convert any control part to data and pass it to the application before passing any data part originally present

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

in the same message. In control-discard mode, *read*( ) shall discard message control parts but return to the process any data part in the message.

In addition, *read*( ) shall fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of *errno* shall not reflect the result of *read*( ), but reflect the prior error. If a hangup occurs on the STREAM being read, *read*( ) shall continue to operate normally until the STREAM head read queue is empty. Thereafter, it shall return 0.

XSI The *pread*( ) function shall be equivalent to *read*( ), except that it shall read from a given position in the file without changing the file pointer. The first three arguments to *pread*( ) are the same as *read*( ) with the addition of a fourth argument *offset* for the desired position inside the file. An attempt to perform a *pread*( ) on a file that is incapable of seeking shall result in an error.

RETURN VALUE

XSI Upon successful completion, *read*( ) and *pread*( ) shall return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions shall return −1 and set *errno* to indicate the error.

ERRORS

XSI The *read*( ) and *pread*( ) functions shall fail if:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be delayed.

[EBADF] The *fildes* argument is not a valid file descriptor open for reading.

XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message waiting to be read includes a control part.

[EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

XSR [EINVAL] The STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

[EIO] The process is a member of a background process attempting to read from its controlling terminal, the process is ignoring or blocking the SIGTTIN signal, or the process group is orphaned. This error may also be generated for implementation-defined reasons.

XSI [EISDIR] The *fildes* argument refers to a directory and the implementation does not allow the directory to be read using *read*( ) or *pread*( ). The *readdir*( ) function should be used instead.

[EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with *fildes*.

The *read*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]
The file descriptor is for a socket, is marked O_NONBLOCK, and no data is waiting to be received.

[ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by its peer.

[ENOTCONN] A read was attempted on a socket that is not connected.

[ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *read*( ) and *pread*( ) functions may fail if:

[EIO] A physical I/O error has occurred.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENOMEM] Insufficient memory was available to fulfill the request.

[ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The *pread*( ) function shall fail, and the file pointer shall remain unchanged, if:

XSI [EINVAL] The *offset* argument is invalid. The value is negative.

XSI [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the file.

XSI [ENXIO] A request was outside the capabilities of the device.

XSI [ESPIPE] *fildes* is associated with a pipe or FIFO.

EXAMPLES

Reading Data into a Buffer

The following example reads data from the file associated with the file descriptor *fd* into the buffer pointed to by *buf*.

```
#include <sys/types.h>
#include <unistd.h>
...
char buf[20];
size_t nbytes;
ssize_t bytes_read;
int fd;
...
nbytes = sizeof(buf);
bytes_read = read(fd, buf, nbytes);
...
```

APPLICATION USAGE

None.

RATIONALE

This volume of IEEE Std 1003.1-2001 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

Note that a *read*( ) of zero bytes does not modify *st_atime*. A *read*( ) that requests more than zero bytes, but returns zero, shall modify *st_atime*.

Implementations are allowed, but not required, to perform error checking for *read*( ) requests of zero bytes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Input and Output

The use of I/O with large byte counts has always presented problems. Ideas such as *lread*( ) and *lwrite*( ) (using and returning **long**s) were considered at one time. The current solution is to use abstract types on the ISO C standard function to *read*( ) and *write*( ). The abstract types can be declared so that existing functions work, but can also be declared so that larger types can be represented in future implementations. It is presumed that whatever constraints limit the maximum range of **size_t** also limit portable I/O requests to the same range. This volume of IEEE Std 1003.1-2001 also limits the range further by requiring that the byte count be limited so that a signed return value remains meaningful. Since the return type is also a (signed) abstract type, the byte count can be defined by the implementation to be larger than an **int** can hold.

The standard developers considered adding atomicity requirements to a pipe or FIFO, but recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of reads of {PIPE_BUF} or any other size that would be an aid to applications portability.

This volume of IEEE Std 1003.1-2001 requires that no action be taken for *read*( ) or *write*( ) when *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid buffer pointers or file descriptors). This is consistent with the rest of this volume of IEEE Std 1003.1-2001, but the phrasing here could be misread to require detection of the zero case before any other errors. A value of zero is to be considered a correct value, for which the semantics are a no-op.

I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the bytes from a single operation that started out together end up together, without interleaving from other I/O operations. It is a known attribute of terminals that this is not honored, and terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified. The behavior for other device types is also left unspecified, but the wording is intended to imply that future standards might choose to specify atomicity (or not).

There were recommendations to add format parameters to *read*( ) and *write*( ) in order to handle networked transfers among heterogeneous file system and base hardware types. Such a facility may be required for support by the OSI presentation of layer services. However, it was determined that this should correspond with similar C-language facilities, and that is beyond the scope of this volume of IEEE Std 1003.1-2001. The concept was suggested to the developers of the ISO C standard for their consideration as a possible area for future work.

In 4.3 BSD, a *read*( ) or *write*( ) that is interrupted by a signal before transferring any data does not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition, there is an additional function, *select*( ), whose purpose is to pause until specified activity (data to read, space to write, and so on) is detected on specified file descriptors. It is common in applications written for those systems for *select*( ) to be used before *read*( ) in situations (such as keyboard input) where interruption of I/O due to a signal is desired.

The issue of which files or file types are interruptible is considered an implementation design issue. This is often affected primarily by hardware and reliability issues.

There are no references to actions taken following an ‘‘unrecoverable error’’. It is considered beyond the scope of this volume of IEEE Std 1003.1-2001 to describe what happens in the case of hardware errors.

Previous versions of IEEE Std 1003.1-2001 allowed two very different behaviors with regard to the handling of interrupts. In order to minimize the resulting confusion, it was decided that IEEE Std 1003.1-2001 should support only one of these behaviors. Historical practice on AT&T-derived systems was to have *read*( ) and *write*( ) return –1 and set *errno* to [EINTR] when interrupted after some, but not all, of the data requested had been transferred. However, the U.S. Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

which *read*( ) and *write*( ) return the number of bytes actually transferred before the interrupt. If –1 is returned when any data is transferred, it is difficult to recover from the error on a seekable device and impossible on a non-seekable device. Most new implementations support this behavior. The behavior required by IEEE Std 1003.1-2001 is to return the number of bytes transferred.

IEEE Std 1003.1-2001 does not specify when an implementation that buffers *read*( )ss actually moves the data into the user-supplied buffer, so an implementation may chose to do this at the latest possible moment. Therefore, an interrupt arriving earlier may not cause *read*( ) to return a partial byte count, but rather to return –1 and set *errno* to [EINTR].

Consideration was also given to combining the two previous options, and setting *errno* to [EINTR] while returning a short count. However, not only is there no existing practice that implements this, it is also contradictory to the idea that when *errno* is set, the function responsible shall return –1.

FUTURE DIRECTIONS

None.

SEE ALSO

fcntl( ), *ioctl*( ), *lseek*( ), *open*( ), *pipe*( ), *readv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<stropts.h>**, **<sys/uio.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

The *pread*( ) function is added.

Issue 6

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now states that if *read*( ) is interrupted by a signal after it has successfully read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read*( ) returned the number of bytes read, or whether it returned –1 with *errno* set to [EINTR]. This is a FIPS requirement.
- In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs past the offset maximum established in the open file description associated with *fildes*. This change is to support large files.
- The [EOVERFLOW] mandatory error condition is added.
- The [ENXIO] optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *read*( ) results are unspecified for typed memory objects.

New RATIONALE is added to explain the atomicity requirements for input and output operations.

The following error conditions are added for operations on sockets: [EAGAIN], [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].

The [EIO] error is changed to ‘‘may fail’’.

The following error conditions are added for operations on sockets: [ENOBUFS] and [ENOMEM].

The *readv*( ) function is split out into a separate reference page.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

readdir, readdir_r — read a directory

SYNOPSIS

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
    struct dirent **restrict result);
```

DESCRIPTION

The type **DIR**, which is defined in the **<dirent.h>** header, represents a *directory stream*, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of *readdir*( ).

The *readdir*( ) function shall return a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument *dirp*, and position the directory stream at the next entry. It shall return a null pointer upon reaching the end of the directory stream. The structure **dirent** defined in the **<dirent.h>** header describes a directory entry.

The *readdir*( ) function shall not return directory entries containing empty names. If entries for dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-dot; otherwise, they shall not be returned.

The pointer returned by *readdir*( ) points to data which may be overwritten by another call to *readdir*( ) on the same directory stream. This data is not overwritten by another call to *readdir*( ) on a different directory stream.

If a file is removed from or added to the directory after the most recent call to *opendir*( ) or *rewinddir*( ), whether a subsequent call to *readdir*( ) returns an entry for that file is unspecified.

The *readdir*( ) function may buffer several directory entries per actual read operation; *readdir*( ) shall mark for update the *st_atime* field of the directory each time the directory is actually read.

After a call to *fork*( ), either the parent or child (but not both) may continue processing the directory stream using *readdir*( ), *rewinddir*( ), or *seekdir*( ). If both the parent and child processes use these functions, the result is undefined.

If the entry names a symbolic link, the value of the *d_ino* member is unspecified.

The *readdir*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

The *readdir_r*( ) function shall initialize the **dirent** structure referenced by *entry* to represent the directory entry at the current position in the directory stream referred to by *dirp*, store a pointer to this structure at the location referenced by *result*, and position the directory stream at the next entry.

The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name* members containing at least {NAME_MAX}+1 elements.

Upon successful return, the pointer returned at **result* shall have the same value as the argument *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

The *readdir_r*( ) function shall not return directory entries containing empty names.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If a file is removed from or added to the directory after the most recent call to *opendir*( ) or *rewinddir*( ), whether a subsequent call to *readdir_r*( ) returns an entry for that file is unspecified.

The *readdir_r*( ) function may buffer several directory entries per actual read operation; the *readdir_r*( ) function shall mark for update the *st_atime* field of the directory each time the directory is actually read.

Applications wishing to check for error situations should set *errno* to 0 before calling *readdir*( ). If *errno* is set to non-zero on return, an error occurred.

RETURN VALUE

Upon successful completion, *readdir*( ) shall return a pointer to an object of type **struct dirent**. When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate the error. When the end of the directory is encountered, a null pointer shall be returned and *errno* is not changed.

TSF If successful, the *readdir_r*( ) function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *readdir*( ) function shall fail if:

[EOVERFLOW] One of the values in the structure to be returned cannot be represented correctly.

The *readdir*( ) function may fail if:

[EBADF] The *dirp* argument does not refer to an open directory stream.

[ENOENT] The current position of the directory stream is invalid.

The *readdir_r*( ) function may fail if:

[EBADF] The *dirp* argument does not refer to an open directory stream.

EXAMPLES

The following sample program searches the current directory for each of the arguments supplied on the command line.

```
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

static void lookup(const char *arg)
{
    DIR *dirp;
    struct dirent *dp;

    if ((dirp = opendir(".")) == NULL) {
        perror("couldn't open '.'");
        return;
    }

    do {
        errno = 0;
        if ((dp = readdir(dirp)) != NULL) {
            if (strcmp(dp->d_name, arg) != 0)
                continue;
```

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                    (void) printf("found %s\n", arg);
                    (void) closedir(dirp);
                        return;
                }
            } while (dp != NULL);
            if (errno != 0)
                perror("error reading directory");
            else
                (void) printf("failed to find %s\n", arg);
            (void) closedir(dirp);
            return;
        }
        int main(int argc, char *argv[])
        {
            int i;
            for (i = 1; i < argc; i++)
                lookup(arvg[i]);
            return (0);
        }
```

APPLICATION USAGE

The *readdir*( ) function should be used in conjunction with *opendir*( ), *closedir*( ), and *rewinddir*( ) to examine the contents of the directory.

The *readdir_r*( ) function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The returned value of *readdir*( ) merely *represents* a directory entry. No equivalence should be inferred.

Historical implementations of *readdir*( ) obtain multiple directory entries on a single read operation, which permits subsequent *readdir*( ) operations to operate from the buffered information. Any wording that required each successful *readdir*( ) operation to mark the directory *st_atime* field for update would disallow such historical performance-oriented implementations.

Since *readdir*( ) returns NULL when it detects an error and when the end of the directory is encountered, an application that needs to tell the difference must set *errno* to zero before the call and check it if NULL is returned. Since the function must not change *errno* in the second case and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL indicates end-of-directory; otherwise, an error.

Routines to deal with this problem more directly were proposed:

```
int derror (dirp)
DIR *dirp;

void clearderr (dirp)
DIR *dirp;
```

The first would indicate whether an error had occurred, and the second would clear the error indication. The simpler method involving *errno* was adopted instead by requiring that *readdir*( ) not change *errno* when end-of-directory is encountered.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

An error or signal indicating that a directory has changed while open was considered but rejected.

The thread-safe version of the directory reading function returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call. Either the {NAME_MAX} compile-time constant or the corresponding *pathconf*( ) option can be used to determine the maximum sizes of returned pathnames.

FUTURE DIRECTIONS

None.

SEE ALSO

closedir( ), *lstat*( ), *opendir*( ), *rewinddir*( ), *symlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 5

Large File Summit extensions are added.

The *readdir_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *readdir*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *readdir_r*( ) function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r*( ).

The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful return for the *readdir_r*( ) function.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- A statement is added to the DESCRIPTION indicating the disposition of certain fields in **struct dirent** when an entry refers to a symbolic link.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [ENOENT] optional error condition is added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *readdir_r*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES section with a new example.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

readlink — read the contents of a symbolic link

SYNOPSIS

```
#include <unistd.h>

ssize_t readlink(const char *restrict path, char *restrict buf,
    size_t bufsize);
```

DESCRIPTION

The *readlink*( ) function shall place the contents of the symbolic link referred to by *path* in the buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*, the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to contain the link content, the first *bufsize* bytes shall be placed in *buf*.

If the value of *bufsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

RETURN VALUE

Upon successful completion, *readlink*( ) shall return the count of bytes placed in the buffer. Otherwise, it shall return a value of –1, leave the buffer unchanged, and set *errno* to indicate the error.

ERRORS

The *readlink*( ) function shall fail if:

[EACCES] Search permission is denied for a component of the path prefix of *path*.

[EINVAL] The *path* argument names a file that is not a symbolic link.

[EIO] An I/O error occurred while reading from the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

The *readlink*( ) function may fail if:

[EACCES] Read permission is denied for the directory.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Reading the Name of a Symbolic Link

The following example shows how to read the name of a symbolic link named **/modules/pass1**.

```
#include <unistd.h>

char buf[1024];
ssizet_t len;
...
if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
    buf[len] = '\0';
```

APPLICATION USAGE

Conforming applications should not assume that the returned contents of the symbolic link are null-terminated.

RATIONALE

Since IEEE Std 1003.1-2001 does not require any association of file times with symbolic links, there is no requirement that file times be updated by *readlink*(). The type associated with *bufsiz* is a **size_t** in order to be consistent with both the ISO C standard and the definition of *read*(). The behavior specified for *readlink*() when *bufsiz* is zero represents historical practice. For this case, the standard developers considered a change whereby *readlink*() would return the number of non-null bytes contained in the symbolic link with the buffer *buf* remaining unchanged; however, since the **stat** structure member *st_size* value can be used to determine the size of buffer necessary to contain the contents of the symbolic link as returned by *readlink*(), this proposal was rejected, and the historical practice retained.

FUTURE DIRECTIONS

None.

SEE ALSO

lstat(), *stat*(), *symlink*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The return type is changed to **ssize_t**, to align with the IEEE P1003.1a draft standard.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- This function is made mandatory.
- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The FUTURE DIRECTIONS section is changed to None.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

36947 The **restrict** keyword is added to the *readlink*() prototype for alignment with the
36948 ISO/IEC 9899:1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

readv — read a vector

SYNOPSIS

XSI
```
#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

DESCRIPTION

The *readv*( ) function shall be equivalent to *read*( ), except as described below. The *readv*( ) function shall place the input data into the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than or equal to {IOV_MAX}.

Each *iovec* entry specifies the base address and length of an area in memory where data should be placed. The *readv*( ) function shall always fill an area completely before proceeding to the next.

Upon successful completion, *readv*( ) shall mark for update the *st_atime* field of the file.

RETURN VALUE

Refer to *read*( ).

ERRORS

Refer to *read*( ).

In addition, the *readv*( ) function shall fail if:

[EINVAL] The sum of the *iov_len* values in the *iov* array overflowed an **ssize_t**.

The *readv*( ) function may fail if:

[EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

EXAMPLES

Reading Data into an Array

The following example reads data from the file associated with the file descriptor *fd* into the buffers specified by members of the *iov* array.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
...
ssize_t bytes_read;
int fd;
char buf0[20];
char buf1[30];
char buf2[40];
int iovcnt;
struct iovec iov[3];

iov[0].iov_base = buf0;
iov[0].iov_len = sizeof(buf0);
iov[1].iov_base = buf1;
iov[1].iov_len = sizeof(buf1);
iov[2].iov_base = buf2;
iov[2].iov_len = sizeof(buf2);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        ...
        iovcnt = sizeof(iov) / sizeof(struct iovec);
        bytes_read = readv(fd, iov, iovcnt);
        ...
```

APPLICATION USAGE

None.

RATIONALE

Refer to *read*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

read( ), *writev*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/uio.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

Split out from the *read*( ) reference page.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

37009 **NAME**

37010 realloc — memory reallocator

37011 **SYNOPSIS**

37012 `#include <stdlib.h>`

37013 `void *realloc(void *`*`ptr`*`, size_t `*`size`*`);`

37014 **DESCRIPTION**

37015 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37016 conflict between the requirements described here and the ISO C standard is unintentional. This
37017 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37018 The *realloc*( ) function shall change the size of the memory object pointed to by *ptr* to the size
37019 specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new
37020 and old sizes. If the new size of the memory object would require movement of the object, the
37021 space for the previous instantiation of the object is freed. If the new size is larger, the contents of
37022 the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,
37023 the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged.

37024 If *ptr* is a null pointer, *realloc*( ) shall be equivalent to *malloc*( ) for the specified size.

37025 If *ptr* does not match a pointer returned earlier by *calloc*( ), *malloc*( ), or *realloc*( ) or if the space has
37026 previously been deallocated by a call to *free*( ) or *realloc*( ), the behavior is undefined.

37027 The order and contiguity of storage allocated by successive calls to *realloc*( ) is unspecified. The
37028 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
37029 a pointer to any type of object and then used to access such an object in the space allocated (until
37030 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
37031 disjoint from any other object. The pointer returned shall point to the start (lowest byte address)
37032 of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

37033 **RETURN VALUE**

37034 Upon successful completion with a size not equal to 0, *realloc*( ) shall return a pointer to the
37035 (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be
37036 successfully passed to *free*( ) shall be returned. If there is not enough available memory, *realloc*( )
37037 CX shall return a null pointer and set *errno* to [ENOMEM].

37038 **ERRORS**

37039 The *realloc*( ) function shall fail if:

37040 CX [ENOMEM] Insufficient memory is available.

37041 **EXAMPLES**

37042 None.

37043 **APPLICATION USAGE**

37044 None.

37045 **RATIONALE**

37046 None.

37047 **FUTURE DIRECTIONS**

37048 None.

37049 **SEE ALSO**

37050 *calloc*( ), *free*( ), *malloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37051 **CHANGE HISTORY**

37052 First released in Issue 1. Derived from Issue 1 of the SVID.

37053 **Issue 6**

37054 Extensions beyond the ISO C standard are marked.

37055 The following new requirements on POSIX implementations derive from alignment with the
37056 Single UNIX Specification:

37057 • In the RETURN VALUE section, if there is not enough available memory, the setting of *errno*
37058 to [ENOMEM] is added.

37059 • The [ENOMEM] error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

realpath — resolve a pathname

SYNOPSIS

XSI
```
#include <stdlib.h>

char *realpath(const char *restrict file_name,
    char *restrict resolved_name);
```

DESCRIPTION

The *realpath*( ) function shall derive, from the pathname pointed to by *file_name*, an absolute pathname that names the same file, whose resolution does not involve `'.'`, `'..'`, or symbolic links. The generated pathname shall be stored as a null-terminated string, up to a maximum of {PATH_MAX} bytes, in the buffer pointed to by *resolved_name*.

If *resolved_name* is a null pointer, the behavior of *realpath*( ) is implementation-defined.

RETURN VALUE

Upon successful completion, *realpath*( ) shall return a pointer to the resolved name. Otherwise, *realpath*( ) shall return a null pointer and set *errno* to indicate the error, and the contents of the buffer pointed to by *resolved_name* are undefined.

ERRORS

The *realpath*( ) function shall fail if:

[EACCES] Read or search permission was denied for a component of *file_name*.

[EINVAL] The *file_name* argument is a null pointer.

[EIO] An error occurred while reading from the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of the *file_name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *file_name* does not name an existing file or *file_name* points to an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

The *realpath*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

[ENOMEM] Insufficient storage space is available.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`--`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

EXAMPLES

Generating an Absolute Pathname

The following example generates an absolute pathname for the file identified by the *symlinkpath* argument. The generated pathname is stored in the *actualpath* array.

```
#include <stdlib.h>
...
char *symlinkpath = "/tmp/symlink/file";
char actualpath [PATH_MAX+1];
char *ptr;

ptr = realpath(symlinkpath, actualpath);
```

APPLICATION USAGE

None.

RATIONALE

Since the maximum pathname length is arbitrary unless {PATH_MAX} is defined, an application generally cannot supply a *resolved_name* buffer with size {{PATH_MAX}+1}.

FUTURE DIRECTIONS

In the future, passing a null pointer to *realpath*( ) for the *resolved_name* argument may be defined to have *realpath*( ) allocate space for the generated pathname.

SEE ALSO

getcwd( ), *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The **restrict** keyword is added to the *realpath*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the DESCRIPTION for the case when *resolved_name* is a null pointer, changing the [EINVAL] error text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

37129 **NAME**

37130 recv — receive a message from a connected socket

37131 **SYNOPSIS**

37132 `#include <sys/socket.h>`

37133 `ssize_t recv(int socket, void *buffer, size_t length, int flags);`

37134 **DESCRIPTION**

37135 The *recv*( ) function shall receive a message from a connection-mode or connectionless-mode
37136 socket. It is normally used with connected sockets because it does not permit the application to
37137 retrieve the source address of received data.

37138 The *recv*( ) function takes the following arguments:

37139 *socket* Specifies the socket file descriptor.

37140 *buffer* Points to a buffer where the message should be stored.

37141 *length* Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

37142 *flags* Specifies the type of message reception. Values of this argument are formed by
37143 logically OR'ing zero or more of the following values:

37144 MSG_PEEK Peeks at an incoming message. The data is treated as unread and
37145 the next *recv*( ) or similar function shall still return this data.

37146 MSG_OOB Requests out-of-band data. The significance and semantics of
37147 out-of-band data are protocol-specific.

37148 MSG_WAITALL On SOCK_STREAM sockets this requests that the function block
37149 until the full amount of data can be returned. The function may
37150 return the smaller amount of data if the socket is a message-
37151 based socket, if a signal is caught, if the connection is
37152 terminated, if MSG_PEEK was specified, or if an error is pending
37153 for the socket.

37154 The *recv*( ) function shall return the length of the message written to the buffer pointed to by the
37155 *buffer* argument. For message-based sockets, such as SOCK_DGRAM and SOCK_SEQPACKET,
37156 the entire message shall be read in a single operation. If a message is too long to fit in the
37157 supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be
37158 discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be
37159 ignored. In this case, data shall be returned to the user as soon as it becomes available, and no
37160 data shall be discarded.

37161 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
37162 message.

37163 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
37164 descriptor, *recv*( ) shall block until a message arrives. If no messages are available at the socket
37165 and O_NONBLOCK is set on the socket's file descriptor, *recv*( ) shall fail and set *errno* to
37166 [EAGAIN] or [EWOULDBLOCK].

37167 **RETURN VALUE**

37168 Upon successful completion, *recv*( ) shall return the length of the message in bytes. If no
37169 messages are available to be received and the peer has performed an orderly shutdown, *recv*( )
37170 shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ERRORS

The *recv*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]
: The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

[EBADF]
: The *socket* argument is not a valid file descriptor.

[ECONNRESET]
: A connection was forcibly closed by a peer.

[EINTR]
: The *recv*( ) function was interrupted by a signal that was caught, before any data was available.

[EINVAL]
: The MSG_OOB flag is set and no out-of-band data is available.

[ENOTCONN]
: A receive is attempted on a connection-mode socket that is not connected.

[ENOTSOCK]
: The *socket* argument does not refer to a socket.

[EOPNOTSUPP]
: The specified flags are not supported for this socket type or protocol.

[ETIMEDOUT]
: The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The *recv*( ) function may fail if:

[EIO]
: An I/O error occurred while reading from or writing to the file system.

[ENOBUFS]
: Insufficient resources were available in the system to perform the operation.

[ENOMEM]
: Insufficient memory was available to fulfill the request.

EXAMPLES

None.

APPLICATION USAGE

The *recv*( ) function is equivalent to *recvfrom*( ) with a zero *address_len* argument, and to *read*( ) if no flags are used.

The *select*( ) and *poll*( ) functions can be used to determine when data is available to be received.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

poll( ), *read*( ), *recvmsg*( ), *recvfrom*( ), *select*( ), *send*( ), *sendmsg*( ), *sendto*( ), *shutdown*( ), *socket*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

recvfrom — receive a message from a socket

SYNOPSIS

```
#include <sys/socket.h>

ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
    int flags, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

DESCRIPTION

The *recvfrom*( ) function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

The *recvfrom*( ) function takes the following arguments:

socket Specifies the socket file descriptor.

buffer Points to the buffer where the message should be stored.

length Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

flags Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:

MSG_PEEK Peeks at an incoming message. The data is treated as unread and the next *recvfrom*( ) or similar function shall still return this data.

MSG_OOB Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

MSG_WAITALL On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

address A null pointer, or points to a **sockaddr** structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.

address_len Specifies the length of the **sockaddr** structure pointed to by the *address* argument.

The *recvfrom*( ) function shall return the length of the message written to the buffer pointed to by the *buffer* argument. For message-based sockets, such as SOCK_RAW, SOCK_DGRAM, and SOCK_SEQPACKET, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes available, and no data shall be discarded.

If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first message.

Not all protocols provide the source address for messages. If the *address* argument is not a null pointer and the protocol provides the source address of messages, the source address of the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37251 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,
37252 and the length of this address shall be stored in the object pointed to by the *address_len*
37253 argument.

37254 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
37255 the stored address shall be truncated.

37256 If the *address* argument is not a null pointer and the protocol does not provide the source address
37257 of messages, the value stored in the object pointed to by *address* is unspecified.

37258 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
37259 descriptor, *recvfrom*( ) shall block until a message arrives. If no messages are available at the
37260 socket and O_NONBLOCK is set on the socket's file descriptor, *recvfrom*( ) shall fail and set *errno*
37261 to [EAGAIN] or [EWOULDBLOCK].

37262 **RETURN VALUE**

37263 Upon successful completion, *recvfrom*( ) shall return the length of the message in bytes. If no
37264 messages are available to be received and the peer has performed an orderly shutdown,
37265 *recvfrom*( ) shall return 0. Otherwise, the function shall return –1 and set *errno* to indicate the
37266 error.

37267 **ERRORS**

37268 The *recvfrom*( ) function shall fail if:

37269 [EAGAIN] or [EWOULDBLOCK]
37270 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
37271 to be received; or MSG_OOB is set and no out-of-band data is available and
37272 either the socket's file descriptor is marked O_NONBLOCK or the socket does
37273 not support blocking to await out-of-band data.

37274 [EBADF] The *socket* argument is not a valid file descriptor.

37275 [ECONNRESET] A connection was forcibly closed by a peer.

37276 [EINTR] A signal interrupted *recvfrom*( ) before any data was available.

37277 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

37278 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

37279 [ENOTSOCK] The *socket* argument does not refer to a socket.

37280 [EOPNOTSUPP] The specified flags are not supported for this socket type.

37281 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
37282 transmission timeout on active connection.

37283 The *recvfrom*( ) function may fail if:

37284 [EIO] An I/O error occurred while reading from or writing to the file system.

37285 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

37286 [ENOMEM] Insufficient memory was available to fulfill the request.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The *select*( ) and *poll*( ) functions can be used to determine when data is available to be received.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

poll( ), *read*( ), *recv*( ), *recvmsg*( ), *select*( ), *send*( ), *sendmsg*( ), *sendto*( ), *shutdown*( ), *socket*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37300 **NAME**

37301 recvmsg — receive a message from a socket

37302 **SYNOPSIS**

37303 `#include <sys/socket.h>`

37304 `ssize_t recvmsg(int socket, struct msghdr *message, int flags);`

37305 **DESCRIPTION**

37306 The *recvmsg*( ) function shall receive a message from a connection-mode or connectionless-mode
37307 socket. It is normally used with connectionless-mode sockets because it permits the application
37308 to retrieve the source address of received data.

37309 The *recvmsg*( ) function takes the following arguments:

37310 *socket* Specifies the socket file descriptor.

37311 *message* Points to a **msghdr** structure, containing both the buffer to store the source
37312 address and the buffers for the incoming message. The length and format of
37313 the address depend on the address family of the socket. The *msg_flags* member
37314 is ignored on input, but may contain meaningful values on output.

37315 *flags* Specifies the type of message reception. Values of this argument are formed
37316 by logically OR'ing zero or more of the following values:

37317 MSG_OOB Requests out-of-band data. The significance and semantics
37318 of out-of-band data are protocol-specific.

37319 MSG_PEEK Peeks at the incoming message.

37320 MSG_WAITALL On SOCK_STREAM sockets this requests that the function
37321 block until the full amount of data can be returned. The
37322 function may return the smaller amount of data if the socket
37323 is a message-based socket, if a signal is caught, if the
37324 connection is terminated, if MSG_PEEK was specified, or if
37325 an error is pending for the socket.

37326 The *recvmsg*( ) function shall receive messages from unconnected or connected sockets and shall
37327 return the length of the message.

37328 The *recvmsg*( ) function shall return the total length of the message. For message-based sockets,
37329 such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single
37330 operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the
37331 *flags* argument, the excess bytes shall be discarded, and MSG_TRUNC shall be set in the
37332 *msg_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK_STREAM,
37333 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it
37334 becomes available, and no data shall be discarded.

37335 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
37336 message.

37337 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
37338 descriptor, *recvmsg*( ) shall block until a message arrives. If no messages are available at the
37339 socket and O_NONBLOCK is set on the socket's file descriptor, the *recvmsg*( ) function shall fail
37340 and set *errno* to [EAGAIN] or [EWOULDBLOCK].

37341 In the **msghdr** structure, the *msg_name* and *msg_namelen* members specify the source address if
37342 the socket is unconnected. If the socket is connected, the *msg_name* and *msg_namelen* members
37343 shall be ignored. The *msg_name* member may be a null pointer if no names are desired or
37344 required. The *msg_iov* and *msg_iovlen* fields are used to specify where the received data shall be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

stored. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Each storage area indicated by *msg_iov* is filled with received data in turn until all of the received data is stored or all of the areas have been filled.

Upon successful completion, the *msg_flags* member of the message header shall be the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:

MSG_EOR	End-of-record was received (if supported by the protocol).
MSG_OOB	Out-of-band data was received.
MSG_TRUNC	Normal data was truncated.
MSG_CTRUNC	Control data was truncated.

RETURN VALUE

Upon successful completion, *recvmsg*( ) shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *recvmsg*( ) shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *recvmsg*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]	The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.
[EBADF]	The *socket* argument is not a valid open file descriptor.
[ECONNRESET]	A connection was forcibly closed by a peer.
[EINTR]	This function was interrupted by a signal before any data was available.
[EINVAL]	The sum of the *iov_len* values overflows a **ssize_t**, or the MSG_OOB flag is set and no out-of-band data is available.
[EMSGSIZE]	The *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less than or equal to 0, or is greater than {IOV_MAX}.
[ENOTCONN]	A receive is attempted on a connection-mode socket that is not connected.
[ENOTSOCK]	The *socket* argument does not refer to a socket.
[EOPNOTSUPP]	The specified flags are not supported for this socket type.
[ETIMEDOUT]	The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The *recvmsg*( ) function may fail if:

[EIO]	An I/O error occurred while reading from or writing to the file system.
[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
[ENOMEM]	Insufficient memory was available to fulfill the request.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37383 **EXAMPLES**

37384 None.

37385 **APPLICATION USAGE**

37386 The *select*( ) and *poll*( ) functions can be used to determine when data is available to be received.

37387 **RATIONALE**

37388 None.

37389 **FUTURE DIRECTIONS**

37390 None.

37391 **SEE ALSO**

37392 *poll*( ), *recv*( ), *recvfrom*( ), *select*( ), *send*( ), *sendmsg*( ), *sendto*( ), *shutdown*( ), *socket*( ), the Base
37393 Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

37394 **CHANGE HISTORY**

37395 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

regcomp, regerror, regexec, regfree — regular expression matching

SYNOPSIS

```
#include <regex.h>

int regcomp(regex_t *restrict preg, const char *restrict pattern,
    int cflags);
size_t regerror(int errcode, const regex_t *restrict preg,
    char *restrict errbuf, size_t errbuf_size);
int regexec(const regex_t *restrict preg, const char *restrict string,
    size_t nmatch, regmatch_t pmatch[restrict], int eflags);
void regfree(regex_t *preg);
```

DESCRIPTION

These functions interpret *basic* and *extended* regular expressions as described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions.

The **regex_t** structure is defined in **<regex.h>** and contains at least the following member:

Member Type	Member Name	Description
`size_t`	`re_nsub`	Number of parenthesized subexpressions.

The **regmatch_t** structure is defined in **<regex.h>** and contains at least the following members:

Member Type	Member Name	Description
regoff_t	*rm_so*	Byte offset from start of *string* to start of substring.
regoff_t	*rm_eo*	Byte offset from start of *string* of the first character after the end of substring.

The *regcomp*( ) function shall compile the regular expression contained in the string pointed to by the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the **<regex.h>** header:

REG_EXTENDED Use Extended Regular Expressions.

REG_ICASE Ignore case in match. (See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions.)

REG_NOSUB Report only success/fail in *regexec*( ).

REG_NEWLINE Change the handling of <newline>s, as described in the text.

The default regular expression type for *pattern* is a Basic Regular Expression. The application can specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

If the REG_NOSUB flag was not set in *cflags*, then *regcomp*( ) shall set *re_nsub* to the number of parenthesized subexpressions (delimited by `"\(\)"` in basic regular expressions or `"()"` in extended regular expressions) found in *pattern*.

The *regexec*( ) function compares the null-terminated string specified by *string* with the compiled regular expression *preg* initialized by a previous call to *regcomp*( ). If it finds a match, *regexec*( ) shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the **<regex.h>** header:

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

REG_NOTBOL The first character of the string pointed to by *string* is not the beginning of the line. Therefore, the circumflex character (`'^'`), when taken as a special character, shall not match the beginning of *string*.

REG_NOTEOL The last character of the string pointed to by *string* is not the end of the line. Therefore, the dollar sign (`'$'`), when taken as a special character, shall not match the end of *string*.

If *nmatch* is 0 or REG_NOSUB was set in the *cflags* argument to *regcomp*( ), then *regexec*( ) shall ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument points to an array with at least *nmatch* elements, and *regexec*( ) shall fill in the elements of that array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions of *pattern*: *pmatch*[*i*].*rm_so* shall be the byte offset of the beginning and *pmatch*[*i*].*rm_eo* shall be one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th matched open parenthesis, counting from 1.) Offsets in *pmatch*[0] identify the substring that corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch*[*nmatch*−1] shall be filled with −1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself counts as a subexpression), then *regexec*( ) shall still do the match, but shall record only the first *nmatch* substrings.

When matching a basic or extended regular expression, any given parenthesized subexpression of *pattern* might participate in the match of several different substrings of *string*, or it might not match any substring even though the pattern as a whole did match. The following rules shall be used to determine which substrings to report in *pmatch* when matching regular expressions:

1. If subexpression *i* in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in *pmatch*[*i*] shall delimit the last such match.

2. If subexpression *i* is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in *pmatch*[*i*] shall be −1. A subexpression does not participate in the match when:

   `'*'` or `"\{\}"` appears immediately after the subexpression in a basic regular expression, or `'*'`, `'?'`, or `"{ }"` appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched 0 times)

   or:

   `'|'` is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.

3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained within any other subexpression that is contained within *j*, and a match of subexpression *j* is reported in *pmatch*[*j*], then the match or non-match of subexpression *i* reported in *pmatch*[*i*] shall be as described in 1. and 2. above, but within the substring reported in *pmatch*[*j*] rather than the whole string. The offsets in *pmatch*[*i*] are still relative to the start of *string*.

4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch*[*j*] are −1, then the pointers in *pmatch*[*i*] shall also be −1.

5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch*[*i*] shall be the byte offset of the character or null terminator immediately following the zero-length string.

If, when *regexec*( ) is called, the locale is different from when the regular expression was compiled, the result is undefined.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary character except as follows:

1. A <newline> in *string* shall not be matched by a period outside a bracket expression or by any form of a non-matching list (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions).
2. A circumflex (`'^'`) in *pattern*, when used to specify expression anchoring (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3.8, BRE Expression Anchoring), shall match the zero-length string immediately after a <newline> in *string*, regardless of the setting of REG_NOTBOL.
3. A dollar sign (`'$'`) in *pattern*, when used to specify expression anchoring, shall match the zero-length string immediately before a <newline> in *string*, regardless of the setting of REG_NOTEOL.

The *regfree*( ) function frees any memory allocated by *regcomp*( ) associated with *preg*.

The following constants are defined as error return values:

REG_NOMATCH	*regexec*( ) failed to match.
REG_BADPAT	Invalid regular expression.
REG_ECOLLATE	Invalid collating element referenced.
REG_ECTYPE	Invalid character class type referenced.
REG_EESCAPE	Trailing `'\'` in pattern.
REG_ESUBREG	Number in `"\digit"` invalid or in error.
REG_EBRACK	`"[]"` imbalance.
REG_EPAREN	`"\(\)"` or `"()"` imbalance.
REG_EBRACE	`"\{\}"` imbalance.
REG_BADBR	Content of `"\{\}"` invalid: not a number, number too large, more than two numbers, first larger than second.
REG_ERANGE	Invalid endpoint in range expression.
REG_ESPACE	Out of memory.
REG_BADRPT	`'?'`, `'*'`, or `'+'` not preceded by valid regular expression.

The *regerror*( ) function provides a mapping from error codes returned by *regcomp*( ) and *regexec*( ) to unspecified printable strings. It generates a string corresponding to the value of the *errcode* argument, which the application shall ensure is the last non-zero value returned by *regcomp*( ) or *regexec*( ) with the given value of *preg*. If *errcode* is not such a value, the content of the generated string is unspecified.

If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec*( ) or *regcomp*( ), the *regerror*( ) still generates an error string corresponding to the value of *errcode*, but it might not be as detailed under some implementations.

If the *errbuf_size* argument is not 0, *regerror*( ) shall place the generated string into the buffer of size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit in the buffer, *regerror*( ) shall truncate the string and null-terminate the result.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If *errbuf_size* is 0, *regerror*( ) shall ignore the *errbuf* argument, and return the size of the buffer needed to hold the generated string.

If the *preg* argument to *regexec*( ) or *regfree*( ) is not a compiled regular expression returned by *regcomp*( ), the result is undefined. A *preg* is no longer treated as a compiled regular expression after it is given to *regfree*( ).

RETURN VALUE

Upon successful completion, the *regcomp*( ) function shall return 0. Otherwise, it shall return an integer value indicating an error as described in **<regex.h>**, and the content of *preg* is undefined. If a code is returned, the interpretation shall be as given in **<regex.h>**.

If *regcomp*( ) detects an invalid RE, it may return REG_BADPAT, or it may return one of the error codes that more precisely describes the error.

Upon successful completion, the *regexec*( ) function shall return 0. Otherwise, it shall return REG_NOMATCH to indicate no match.

Upon successful completion, the *regerror*( ) function shall return the number of bytes needed to hold the entire generated string, including the null termination. If the return value is greater than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

The *regfree*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

```
#include <regex.h>

/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * Return 1 for match, 0 for no match.
 */

int
match(const char *string, char *pattern)
{
    int    status;
    regex_t    re;

    if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
        return(0);      /* Report error. */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0);      /* Report error. */
    }
    return(1);
}
```

The following demonstrates how the REG_NOTBOL flag could be used with *regexec*( ) to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
(void) regcomp (&re, pattern, 0);
/* This call to regexec() finds the first match on the line. */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) {  /* While matches found. */
    /* Substring found between pm.rm_so and pm.rm_eo. */
    /* This call to regexec() finds the next match. */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}
```

APPLICATION USAGE

An application could use:

```
regerror(code,preg,(char *)NULL,(size_t)0)
```

to find out how big a buffer is needed for the generated string, *malloc*( ) a buffer to hold the string, and then call *regerror*( ) again to get the string. Alternatively, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use *malloc*( ) to allocate a larger buffer if it finds that this is too small.

To match a pattern as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern Matching Notation, use the *fnmatch*( ) function.

RATIONALE

The *regexec*( ) function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern*. The application writer should note that there is probably no reason for using a value of *nmatch* that is larger than *preg–>re_nsub*+1.

The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like text editors. In such applications, the user supplies an RE asking the application to find a line that matches the given expression. An anchor in such an RE anchors at the beginning or end of any line. Such an application can pass a sequence of <newline>-separated lines to *regexec*( ) as a single long string and specify REG_NEWLINE to *regcomp*( ) to get the desired behavior. The application must ensure that there are no explicit <newline>s in *pattern* if it wants to ensure that any match occurs entirely within a single line.

The REG_NEWLINE flag affects the behavior of *regexec*( ), but it is in the *cflags* parameter to *regcomp*( ) to allow flexibility of implementation. Some implementations will want to generate the same compiled RE in *regcomp*( ) regardless of the setting of REG_NEWLINE and have *regexec*( ) handle anchors differently based on the setting of the flag. Other implementations will generate different compiled REs based on the REG_NEWLINE.

The REG_ICASE flag supports the operations taken by the *grep* **–i** option and the historical implementations of *ex* and *vi*. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch*[ ] are defined using offsets from the start of the string rather than pointers. Since this is a new interface, there should be no impact on historical implementations or applications, and offsets should be just as easy to use as pointers. The change to offsets was made to facilitate future extensions in which the string to be searched is presented to *regexec*( ) in blocks, allowing a string to be searched that is not all in memory at once.

The type **regoff_t** is used for the elements of *pmatch*[ ] to ensure that the application can represent either the largest possible array in memory (important for an application conforming to the Shell and Utilities volume of IEEE Std 1003.1-2001) or the largest possible file (important for an application using the extension where a file is searched in chunks).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

The standard developers rejected the inclusion of a *regsub*( ) function that would be used to do substitutions for a matched RE. While such a routine would be useful to some applications, its utility would be much more limited than the matching function described here. Both RE parsing and substitution are possible to implement without support other than that required by the ISO C standard, but matching is much more complex than substituting. The only difficult part of substitution, given the information supplied by *regexec*( ), is finding the next character in a string when there can be multi-byte characters. That is a much larger issue, and one that needs a more general solution.

The *errno* variable has not been used for error returns to avoid filling the *errno* name space for this feature.

The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used simultaneously in several contexts; in *main*( ) and a signal handler, perhaps, or in multiple threads of lightweight processes. (The *preg* argument to *regexec*( ) is declared with type **const**, so the implementation is not permitted to use the structure to store intermediate results.) It also allows an application to request an arbitrary number of substrings from an RE. The number of subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexec*( ), consideration was given to dropping the REG_NOSUB flag since the user can now specify this with a zero *nmatch* argument to *regexec*( ). However, keeping REG_NOSUB allows an implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp*( ) that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in the ISO C standard, is unsigned, so the description of *regexec*( ) does not need to address negative values of *nmatch*.

REG_NOTBOL was added to allow an application to do repeated searches for the same pattern in a line. If the pattern contains a circumflex character that should match the beginning of a line, then the pattern should only match when matched against the beginning of the line. Without the REG_NOTBOL flag, the application could rewrite the expression for subsequent matches, but in the general case this would require parsing the expression. The need for REG_NOTEOL is not as clear; it was added for symmetry.

The addition of the *regerror*( ) function addresses the historical need for conforming application programs to have access to error information more than ''Function failed to compile/match your RE for unknown reasons''.

This interface provides for two different methods of dealing with error conditions. The specific error codes (REG_EBRACE, for example), defined in **<regex.h>**, allow an application to recover from an error if it is so able. Many applications, especially those that use patterns supplied by a user, will not try to deal with specific error cases, but will just use *regerror*( ) to obtain a human-readable error message to present to the user.

The *regerror*( ) function uses a scheme similar to *confstr*( ) to deal with the problem of allocating memory to hold the generated string. The scheme used by *strerror*( ) in the ISO C standard was considered unacceptable since it creates difficulties for multi-threaded applications.

The *preg* argument is provided to *regerror*( ) to allow an implementation to generate a more descriptive message than would be possible with *errcode* alone. An implementation might, for example, save the character offset of the offending character of the pattern in a field of *preg*, and then include that in the generated message string. The implementation may also ignore *preg*.

A REG_FILENAME flag was considered, but omitted. This flag caused *regexec*( ) to match patterns as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern Matching Notation instead of REs. This service is now provided by the *fnmatch*( )

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

function.

Notice that there is a difference in philosophy between the ISO POSIX-2: 1993 standard and IEEE Std 1003.1-2001 in how to handle a ''bad'' regular expression. The ISO POSIX-2: 1993 standard says that many bad constructs ''produce undefined results'', or that ''the interpretation is undefined''. IEEE Std 1003.1-2001, however, says that the interpretation of such REs is unspecified. The term ''undefined'' means that the action by the application is an error, of similar severity to passing a bad pointer to a function.

The *regcomp*( ) and *regexec*( ) functions are required to accept any null-terminated string as the *pattern* argument. If the meaning of the string is ''undefined'', the behavior of the function is ''unspecified''. IEEE Std 1003.1-2001 does not specify how the functions will interpret the pattern; they might return error codes, or they might do pattern matching in some completely unexpected way, but they should not do something like abort the process.

FUTURE DIRECTIONS

None.

SEE ALSO

fnmatch( ), *glob*( ), Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern Matching Notation, Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions, **<regex.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

The REG_ENOSYS constant is removed.

The **restrict** keyword is added to the *regcomp*( ), *regerror*( ), and *regexec*( ) prototypes for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

remainder, remainderf, remainderl — remainder function

SYNOPSIS

```
#include <math.h>

double remainder(double x, double y);
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall return the floating-point remainder $r=x-ny$ when y is non-zero. The value n is the integral value nearest the exact value x/y. When $|n-x/y|=\frac{1}{2}$, the value n is chosen to be even.

The behavior of *remainder*( ) shall be independent of the rounding mode.

RETURN VALUE

Upon successful completion, these functions shall return the floating-point remainder $r=x-ny$ when y is non-zero.

MX If x or y is NaN, a NaN shall be returned.

If x is infinite or y is 0 and the other is non-NaN, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The x argument is ±Inf, or the y argument is ±0 and the other argument is non-NaN.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abs( ), *div*( ), *feclearexcept*( ), *fetestexcept*( ), *ldiv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

37740 **CHANGE HISTORY**

37741 First released in Issue 4, Version 2.

37742 **Issue 5**

37743 Moved from X/OPEN UNIX extension to BASE.

37744 **Issue 6**

37745 The *remainder*( ) function is no longer marked as an extension.

37746 The *remainderf*( ) and *remainderl*( ) functions are added for alignment with the ISO/IEC 9899:1999
37747 standard.

37748 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
37749 revised to align with the ISO/IEC 9899:1999 standard.

37750 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
37751 marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

remove — remove a file

SYNOPSIS

```
#include <stdio.h>

int remove(const char *path);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *remove*( ) function shall cause the file named by the pathname pointed to by *path* to be no longer accessible by that name. A subsequent attempt to open that file using that name shall fail, unless it is created anew.

CX If *path* does not name a directory, *remove*(*path*) shall be equivalent to *unlink*(*path*).

If *path* names a directory, *remove*(*path*) shall be equivalent to *rmdir*(*path*).

RETURN VALUE

CX Refer to *rmdir*( ) or *unlink*( ).

ERRORS

CX Refer to *rmdir*( ) or *unlink*( ).

EXAMPLES

Removing Access to a File

The following example shows how to remove access to a file named **/home/cnd/old_mods**.

```
#include <stdio.h>

int status;
...
status = remove("/home/cnd/old_mods");
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

rmdir( ), *unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove*() is equivalent to *unlink*(), and if it is a directory, it is equivalent to *rmdir*().

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37795 **NAME**

37796 remque — remove an element from a queue

37797 **SYNOPSIS**

37798 XSI
```
#include <search.h>

void remque(void *element);
```
37799

37800

37801 **DESCRIPTION**

37802 Refer to *insque*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

remquo, remquof, remquol — remainder functions

SYNOPSIS

```
#include <math.h>

double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *remquo*( ), *remquof*( ), and *remquol*( ) functions shall compute the same remainder as the *remainder*( ), *remainderf*( ), and *remainderl*( ) functions, respectively. In the object pointed to by *quo*, they store a value whose sign is the sign of x/y and whose magnitude is congruent modulo 2^n to the magnitude of the integral quotient of x/y, where n is an implementation-defined integer greater than or equal to 3.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

These functions shall return *x* REM *y*.

MX If *x* or *y* is NaN, a NaN shall be returned.

If *x* is ±Inf or *y* is zero and the other argument is non-NaN, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is ±Inf, or the *y* argument is ±0 and the other argument is non-NaN.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions are intended for implementing argument reductions which can exploit a few low-order bits of the quotient. Note that *x* may be so large in magnitude relative to *y* that an exact representation of the quotient is not practical.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37845 **FUTURE DIRECTIONS**

37846 None.

37847 **SEE ALSO**

37848 *feclearexcept*( ), *fetestexcept*( ), *remainder*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
37849 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

37850 **CHANGE HISTORY**

37851 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

rename — rename a file

SYNOPSIS

```
#include <stdio.h>

int rename(const char *old, const char *new);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *rename*( ) function shall change the name of a file. The *old* argument points to the pathname of the file to be renamed. The *new* argument points to the new pathname of the file.

CX If either the *old* or *new* argument names a symbolic link, *rename*( ) shall operate on the symbolic link itself, and shall not resolve the last component of the argument. If the *old* argument and the *new* argument resolve to the same existing file, *rename*( ) shall return successfully and perform no other action.

If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other processes throughout the renaming operation and refer either to the file referred to by *new* or *old* before the operation began. Write access permission is required for both the directory containing *old* and the directory containing *new*.

If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the pathname of a file that is not a directory. If the directory named by the *new* argument exists, it shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout the renaming operation and shall refer either to the directory referred to by *new* or *old* before the operation began. If *new* names an existing directory, it shall be required to be an empty directory.

If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed. If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.

The *new* pathname shall not contain a path prefix that names *old*. Write access permission is required for the directory containing *old* and the directory containing *new*. If the *old* argument points to the pathname of a directory, write access permission may be required for the directory named by *old*, and, if it exists, the directory named by *new*.

If the link named by the *new* argument exists and the file's link count becomes 0 when it is removed and no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have the file open when the last link is removed, the link shall be removed before *rename*( ) returns, but the removal of the file contents shall be postponed until all references to the file are closed.

Upon successful completion, *rename*( ) shall mark for update the *st_ctime* and *st_mtime* fields of the parent directory of each file.

If the *rename*( ) function fails for any reason other than [EIO], any file named by *new* shall be unaffected.

RETURN VALUE

CX Upon successful completion, *rename*( ) shall return 0; otherwise, –1 shall be returned, *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be changed or created.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37897 **ERRORS**

37898 The *rename*( ) function shall fail if:

37899 CX [EACCES] A component of either path prefix denies search permission; or one of the
37900 directories containing *old* or *new* denies write permissions; or, write
37901 permission is required and is denied for a directory pointed to by the *old* or
37902 *new* arguments.

37903 CX [EBUSY] The directory named by *old* or *new* is currently in use by the system or another
37904 process, and the implementation considers this an error.

37905 CX [EEXIST] or [ENOTEMPTY]
37906 The link named by *new* is a directory that is not an empty directory.

37907 CX [EINVAL] The *new* directory pathname contains a path prefix that names the *old*
37908 directory.

37909 CX [EIO] A physical I/O error has occurred.

37910 CX [EISDIR] The *new* argument points to a directory and the *old* argument points to a file
37911 that is not a directory.

37912 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
37913 argument.

37914 CX [EMLINK] The file named by *old* is a directory, and the link count of the parent directory
37915 of *new* would exceed {LINK_MAX}.

37916 CX [ENAMETOOLONG]
37917 The length of the *old* or *new* argument exceeds {PATH_MAX} or a pathname
37918 component is longer than {NAME_MAX}.

37919 CX [ENOENT] The link named by *old* does not name an existing file, or either *old* or *new*
37920 points to an empty string.

37921 CX [ENOSPC] The directory that would contain *new* cannot be extended.

37922 CX [ENOTDIR] A component of either path prefix is not a directory; or the *old* argument
37923 names a directory and *new* argument names a non-directory file.

37924 XSI [EPERM] or [EACCES]
37925 The S_ISVTX flag is set on the directory containing the file referred to by *old*
37926 and the caller is not the file owner, nor is the caller the directory owner, nor
37927 does the caller have appropriate privileges; or *new* refers to an existing file, the
37928 S_ISVTX flag is set on the directory containing this file, and the caller is not
37929 the file owner, nor is the caller the directory owner, nor does the caller have
37930 appropriate privileges.

37931 CX [EROFS] The requested operation requires writing in a directory on a read-only file
37932 system.

37933 CX [EXDEV] The links named by *new* and *old* are on different file systems and the
37934 implementation does not support links between file systems.

37935 The *rename*( ) function may fail if:

37936 XSI [EBUSY] The file named by the *old* or *new* arguments is a named STREAM.

37937 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
37938 resolution of the *path* argument.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

37939 CX [ENAMETOOLONG]
37940 As a result of encountering a symbolic link in resolution of the *path* argument,
37941 the length of the substituted pathname string exceeded {PATH_MAX}.

37942 CX [ETXTBSY] The file to be renamed is a pure procedure (shared text) file that is being
37943 executed.

37944 **EXAMPLES**

37945 **Renaming a File**

37946 The following example shows how to rename a file named **/home/cnd/mod1** to
37947 **/home/cnd/mod2**.

37948
```
#include <stdio.h>
```

37949
```
int status;
```
37950
```
...
```
37951
```
status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

37952 **APPLICATION USAGE**

37953 Some implementations mark for update the *st_ctime* field of renamed files and some do not.
37954 Applications which make use of the *st_ctime* field may behave differently with respect to
37955 renamed files unless they are designed to allow for either behavior.

37956 **RATIONALE**

37957 This *rename*( ) function is equivalent for regular files to that defined by the ISO C standard. Its
37958 inclusion here expands that definition to include actions on directories and specifies behavior
37959 when the *new* parameter names a file that already exists. That specification requires that the
37960 action of the function be atomic.

37961 One of the reasons for introducing this function was to have a means of renaming directories
37962 while permitting implementations to prohibit the use of *link*( ) and *unlink*( ) with directories,
37963 thus constraining links to directories to those made by *mkdir*( ).

37964 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

37965
```
rename("x", "x");
```

37966 does not remove the file.

37967 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

37968 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir*( ) and [EBUSY] in
37969 *unlink*( ). For a discussion of [EXDEV], see *link*( ).

37970 **FUTURE DIRECTIONS**

37971 None.

37972 **SEE ALSO**

37973 *link*( ), *rmdir*( ), *symlink*( ), *unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
37974 **<stdio.h>**

37975 **CHANGE HISTORY**

37976 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

37977 **Issue 5**

37978 The [EBUSY] error is added to the ‘‘may fail’’ part of the ERRORS section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

rewind — reset the file position indicator in a stream

SYNOPSIS

```
#include <stdio.h>

void rewind(FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The call:

```
rewind(stream)
```

shall be equivalent to:

```
(void) fseek(stream, 0L, SEEK_SET)
```

except that *rewind*( ) shall also clear the error indicator.

CX Since *rewind*( ) does not return a value, an application wishing to detect errors should clear *errno*, then call *rewind*( ), and if *errno* is non-zero, assume an error has occurred.

RETURN VALUE

The *rewind*( ) function shall not return a value.

ERRORS

CX Refer to *fseek*( ) with the exception of [EINVAL] which does not apply.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fseek( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38025 **NAME**

38026 rewinddir — reset the position of a directory stream to the beginning of a directory

38027 **SYNOPSIS**

38028 `#include <dirent.h>`

38029 `void rewinddir(DIR *dirp);`

38030 **DESCRIPTION**

38031 The *rewinddir*( ) function shall reset the position of the directory stream to which *dirp* refers to
38032 the beginning of the directory. It shall also cause the directory stream to refer to the current state
38033 of the corresponding directory, as a call to *opendir*( ) would have done. If *dirp* does not refer to a
38034 directory stream, the effect is undefined.

38035 After a call to the *fork*( ) function, either the parent or child (but not both) may continue
38036 XSI processing the directory stream using *readdir*( ), *rewinddir*( ), or *seekdir*( ). If both the parent and
38037 child processes use these functions, the result is undefined.

38038 **RETURN VALUE**

38039 The *rewinddir*( ) function shall not return a value.

38040 **ERRORS**

38041 No errors are defined.

38042 **EXAMPLES**

38043 None.

38044 **APPLICATION USAGE**

38045 The *rewinddir*( ) function should be used in conjunction with *opendir*( ), *readdir*( ), and *closedir*( ) to
38046 examine the contents of the directory. This method is recommended for portability.

38047 **RATIONALE**

38048 None.

38049 **FUTURE DIRECTIONS**

38050 None.

38051 **SEE ALSO**

38052 *closedir*( ), *opendir*( ), *readdir*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**
38053 **<sys/types.h>**

38054 **CHANGE HISTORY**

38055 First released in Issue 2.

38056 **Issue 6**

38057 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

38058 The following new requirements on POSIX implementations derive from alignment with the
38059 Single UNIX Specification:

38060 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
38061 required for conforming implementations of previous POSIX specifications, it was not
38062 required for UNIX applications.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38063 **NAME**

38064 rindex — character string operations (**LEGACY**)

38065 **SYNOPSIS**

38066 XSI
```
#include <strings.h>
```

38067
```
char *rindex(const char *s, int c);
```

38068

38069 **DESCRIPTION**

38070 The *rindex*( ) function shall be equivalent to *strrchr*( ).

38071 **RETURN VALUE**

38072 Refer to *strrchr*( ).

38073 **ERRORS**

38074 Refer to *strrchr*( ).

38075 **EXAMPLES**

38076 None.

38077 **APPLICATION USAGE**

38078 The *strrchr*( ) function is preferred over this function.

38079 For maximum portability, it is recommended to replace the function call to *rindex*( ) as follows:

38080
```
#define rindex(a,b) strrchr((a),(b))
```

38081 **RATIONALE**

38082 None.

38083 **FUTURE DIRECTIONS**

38084 This function may be withdrawn in a future version.

38085 **SEE ALSO**

38086 *strrchr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

38087 **CHANGE HISTORY**

38088 First released in Issue 4, Version 2.

38089 **Issue 5**

38090 Moved from X/OPEN UNIX extension to BASE.

38091 **Issue 6**

38092 This function is marked LEGACY.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

rint, rintf, rintl — round-to-nearest integral value

SYNOPSIS

```
#include <math.h>

double rint(double x);
float rintf(float x);
long double rintl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall return the integral value (represented as a **double**) nearest *x* in the direction of the current rounding mode. The current rounding mode is implementation-defined.

If the current rounding mode rounds toward negative infinity, then *rint*( ) shall be equivalent to *floor*( ). If the current rounding mode rounds toward positive infinity, then *rint*( ) shall be equivalent to *ceil*( ).

These functions differ from the *nearbyint*( ), *nearbyintf*( ), and *nearbyintl*( ) functions only in that they may raise the inexact floating-point exception if the result differs in value from the argument.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the integer (represented as a double precision number) nearest *x* in the direction of the current rounding mode.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

XSI If the correct value would cause overflow, a range error shall occur and *rint*( ), *rintf*( ), and *rintl*( ) shall return the value of the macro ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*), respectively.

ERRORS

These functions shall fail if:

XSI

Range Error	The result would cause an overflow.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38131 **EXAMPLES**

38132 None.

38133 **APPLICATION USAGE**

38134 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
38135 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38136 **RATIONALE**

38137 None.

38138 **FUTURE DIRECTIONS**

38139 None.

38140 **SEE ALSO**

38141 *abs*( ), *ceil*( ), *feclearexcept*( ), *fetestexcept*( ), *floor*( ), *isnan*( ), *nearbyint*( ), the Base Definitions volume
38142 of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
38143 **<math.h>**

38144 **CHANGE HISTORY**

38145 First released in Issue 4, Version 2.

38146 **Issue 5**

38147 Moved from X/OPEN UNIX extension to BASE.

38148 **Issue 6**

38149 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

38150 - The *rintf*( ) and *rintl*( ) functions are added.

38151 - The *rint*( ) function is no longer marked as an extension.

38152 - The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
38153 revised to align with the ISO/IEC 9899:1999 standard.

38154 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
38155 marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

rmdir — remove a directory

SYNOPSIS

```
#include <unistd.h>

int rmdir(const char *path);
```

DESCRIPTION

The *rmdir*( ) function shall remove a directory whose name is given by *path*. The directory shall be removed only if it is an empty directory.

If the directory is the root directory or the current working directory of any process, it is unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].

If *path* names a symbolic link, then *rmdir*( ) shall fail and set *errno* to [ENOTDIR].

If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir*( ) shall fail.

If the directory's link count becomes 0 and no process has the directory open, the space occupied by the directory shall be freed and the directory shall no longer be accessible. If one or more processes have the directory open when the last link is removed, the dot and dot-dot entries, if present, shall be removed before *rmdir*( ) returns and no new entries may be created in the directory, but the directory shall not be removed until all references to the directory are closed.

If the directory is not an empty directory, *rmdir*( ) shall fail and set *errno* to [EEXIST] or [ENOTEMPTY].

Upon successful completion, the *rmdir*( ) function shall mark for update the *st_ctime* and *st_mtime* fields of the parent directory.

RETURN VALUE

Upon successful completion, the function *rmdir*( ) shall return 0. Otherwise, –1 shall be returned, and *errno* set to indicate the error. If –1 is returned, the named directory shall not be changed.

ERRORS

The *rmdir*( ) function shall fail if:

[EACCES] Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.

[EBUSY] The directory to be removed is currently in use by the system or some process and the implementation considers this to be an error.

[EEXIST] or [ENOTEMPTY]
The *path* argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.

[EINVAL] The *path* argument contains a last component that is dot.

[EIO] A physical I/O error has occurred.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file, or the *path* argument names a nonexistent directory or points to an empty string.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[ENOTDIR] A component of *path* is not a directory.

XSI [EPERM] or [EACCES]
The S_ISVTX flag is set on the parent directory of the directory to be removed and the caller is not the owner of the directory to be removed, nor is the caller the owner of the parent directory, nor does the caller have the appropriate privileges.

[EROFS] The directory entry to be removed resides on a read-only file system.

The *rmdir*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

EXAMPLES

Removing a Directory

The following example shows how to remove a directory named **/home/cnd/mod1**.

```
#include <unistd.h>

int status;
...
status = rmdir("/home/cnd/mod1");
```

APPLICATION USAGE

None.

RATIONALE

The *rmdir*( ) and *rename*( ) functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the condition when the directory to be removed does not exist or *new* already exists. When the 1984 /usr/group standard was published, it contained [EEXIST] instead. When these functions were adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several existing applications and implementations support/use both forms, and no agreement could be reached on either value. All implementations are required to supply both [EEXIST] and [ENOTEMPTY] in **<errno.h>** with distinct values, so that applications can use both values in C-language **case** statements.

The meaning of deleting *pathname*/**dot** is unclear, because the name of the file (directory) in the parent directory to be removed is not clear, particularly in the presence of multiple links to a directory.

The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir*( ) when there are multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or [ENOTEMPTY] clarifies the behavior in this case.

If the process' current working directory is being removed, that should be an allowed error.

Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in Section 2.3 (on page 21) about returning any one of the possible errors permits that behavior to continue. The [ELOOP] error may be returned if more than {SYMLOOP_MAX} symbolic links are encountered during resolution of the *path* argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.3 (on page 21), *mkdir*( ), *remove*( ), *unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38257 **NAME**

38258 round, roundf, roundl — round to the nearest integer value in a floating-point format

38259 **SYNOPSIS**

38260
```
#include <math.h>
```

38261
```
double round(double x);
```
38262
```
float roundf(float x);
```
38263
```
long double roundl(long double x);
```

38264 **DESCRIPTION**

38265 CX The functionality described on this reference page is aligned with the ISO C standard. Any
38266 conflict between the requirements described here and the ISO C standard is unintentional. This
38267 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38268 These functions shall round their argument to the nearest integer value in floating-point format,
38269 rounding halfway cases away from zero, regardless of the current rounding direction.

38270 An application wishing to check for error situations should set *errno* to zero and call
38271 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
38272 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
38273 zero, an error has occurred.

38274 **RETURN VALUE**

38275 Upon successful completion, these functions shall return the rounded integer value.

38276 MX If *x* is NaN, a NaN shall be returned.

38277 If *x* is ±0 or ±Inf, *x* shall be returned.

38278 XSI If the correct value would cause overflow, a range error shall occur and *round*( ), *roundf*( ), and
38279 *roundl*( ) shall return the value of the macro ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL
38280 (with the same sign as *x*), respectively.

38281 **ERRORS**

38282 These functions may fail if:

38283 XSI

Range Error	The result overflows.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

38288 **EXAMPLES**

38289 None.

38290 **APPLICATION USAGE**

38291 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
38292 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38293 **RATIONALE**

38294 None.

38295 **FUTURE DIRECTIONS**

38296 None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

38297 **SEE ALSO**

38298 *feclearexcept*( ), *fetestexcept*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,
38299 Treatment of Error Conditions for Mathematical Functions, **<math.h>**

38300 **CHANGE HISTORY**

38301 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38302 **NAME**

38303 scalb — load exponent of a radix-independent floating-point number

38304 **SYNOPSIS**

38305 OB XSI
```
#include <math.h>

double scalb(double x, double n);
```
38306

38307

38308 **DESCRIPTION**

38309 The *scalb*( ) function shall compute $x*r^n$, where *r* is the radix of the machine's floating-point
38310 arithmetic. When *r* is 2, *scalb*( ) shall be equivalent to *ldexp*( ). The value of *r* is FLT_RADIX
38311 which is defined in **<float.h>**.

38312 An application wishing to check for error situations should set *errno* to zero and call
38313 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
38314 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
38315 zero, an error has occurred.

38316 **RETURN VALUE**

38317 Upon successful completion, the *scalb*( ) function shall return $x*r^n$.

38318 If *x* or *n* is NaN, a NaN shall be returned.

38319 If *n* is zero, *x* shall be returned.

38320 If *x* is ±Inf and *n* is not –Inf, *x* shall be returned.

38321 If *x* is ±0 and *n* is not +Inf, *x* shall be returned.

38322 If *x* is ±0 and *n* is +Inf, a domain error shall occur, and either a NaN (if supported), or an
38323 implementation-defined value shall be returned.

38324 If *x* is ±Inf and *n* is –Inf, a domain error shall occur, and either a NaN (if supported), or an
38325 implementation-defined value shall be returned.

38326 If the result would cause an overflow, a range error shall occur and ±HUGE_VAL (according to
38327 the sign of *x*) shall be returned.

38328 If the correct value would cause underflow, and is representable, a range error may occur and
38329 the correct value shall be returned.

38330 If the correct value would cause underflow, and is not representable, a range error may occur,
38331 and 0.0 shall be returned.

38332 **ERRORS**

38333 The *scalb*( ) function shall fail if:

38334 Domain Error — If *x* is zero and *n* is +Inf, or *x* is Inf and *n* is –Inf.

38335 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
38336 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
38337 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
38338 shall be raised.

38339 Range Error — The result would overflow.

38340 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
38341 then *errno* shall be set to [ERANGE]. If the integer expression
38342 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
38343 floating-point exception shall be raised.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *scalb*( ) function may fail if:

Range Error — The result underflows.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

Applications should use either *scalbln*( ), *scalblnf*( ), or *scalblnl*( ) in preference to this function.

IEEE Std 1003.1-2001 only defines the behavior for the *scalb*( ) function when the *n* argument is an integer, a NaN, or Inf. The behavior of other values for the *n* argument is unspecified.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *ilogb*( ), *ldexp*( ), *logb*( ), *scalbln*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<float.h>**, **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

This function is marked obsolescent.

Although this function is not part of the ISO/IEC 9899:1999 standard, the RETURN VALUE and ERRORS sections are updated to align with the error handling in the ISO/IEC 9899:1999 standard.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38377 **NAME**

38378 scalbln, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT_RADIX

38379 **SYNOPSIS**

38380
```
#include <math.h>

double scalbln(double x, long n);
float scalblnf(float x, long n);
long double scalblnl(long double x, long n);
double scalbn(double x, int n);
float scalbnf(float x, int n);
long double scalbnl(long double x, int n);
```

38387 **DESCRIPTION**

38388 CX The functionality described on this reference page is aligned with the ISO C standard. Any
38389 conflict between the requirements described here and the ISO C standard is unintentional. This
38390 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38391 These functions shall compute $x * \text{FLT_RADIX}^n$ efficiently, not normally by computing
38392 FLT_RADIX^n explicitly.

38393 An application wishing to check for error situations should set *errno* to zero and call
38394 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
38395 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
38396 zero, an error has occurred.

38397 **RETURN VALUE**

38398 Upon successful completion, these functions shall return $x * \text{FLT_RADIX}^n$.

38399 If the result would cause overflow, a range error shall occur and these functions shall return
38400 ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*) as appropriate for
38401 the return type of the function.

38402 If the correct value would cause underflow, and is not representable, a range error may occur,
38403 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

38404 MX If *x* is NaN, a NaN shall be returned.

38405 If *x* is ±0 or ±Inf, *x* shall be returned.

38406 If *n* is 0, *x* shall be returned.

38407 If the correct value would cause underflow, and is representable, a range error may occur and
38408 the correct value shall be returned.

38409 **ERRORS**

38410 These functions shall fail if:

38411 Range Error The result overflows.

38412 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
38413 then *errno* shall be set to [ERANGE]. If the integer expression
38414 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
38415 floating-point exception shall be raised.

38416 These functions may fail if:

38417 Range Error The result underflows.

38418 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
38419 then *errno* shall be set to [ERANGE]. If the integer expression

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions are named so as to avoid conflicting with the historical definition of the *scalb*( ) function from the Single UNIX Specification. The difference is that the *scalb*( ) function has a second argument of **double** instead of **int**. The *scalb*( ) function is not part of the ISO C standard.

The three functions whose second type is **long** are provided because the factor required to scale from the smallest positive floating-point value to the largest finite one, on many implementations, is too large to represent in the minimum-width **int** format.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *scalb*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38441 **NAME**

38442 scanf — convert formatted input

38443 **SYNOPSIS**

38444 `#include <stdio.h>`

38445 `int scanf(const char *restrict format, ... );`

38446 **DESCRIPTION**

38447 Refer to *fscanf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sched_get_priority_max, sched_get_priority_min — get priority limits (**REALTIME**)

SYNOPSIS

PS|TPS

```
#include <sched.h>

int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
```

DESCRIPTION

The *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions shall return the appropriate maximum or minimum, respectively, for the scheduling policy specified by *policy*.

The value of *policy* shall be one of the scheduling policy values defined in **<sched.h>**.

RETURN VALUE

If successful, the *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions shall return the appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions shall fail if:

[EINVAL] The value of the *policy* parameter does not represent a defined scheduling policy.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getparam( ), *sched_setparam*( ), *sched_getscheduler*( ), *sched_rr_get_interval*( ), *sched_setscheduler*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

These functions are marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

The [ESRCH] error condition has been removed since these functions do not take a *pid* argument.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin code in the SYNOPSIS to PS|TPS.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sched_getparam — get scheduling parameters (**REALTIME**)

SYNOPSIS

PS
```
#include <sched.h>

int sched_getparam(pid_t pid, struct sched_param *param);
```

DESCRIPTION

The *sched_getparam*( ) function shall return the scheduling parameters of a process specified by *pid* in the **sched_param** structure pointed to by *param*.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters for the process whose process ID is equal to *pid* shall be returned.

If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of the *sched_getparam*( ) function is unspecified if the value of *pid* is negative.

RETURN VALUE

Upon successful completion, the *sched_getparam*( ) function shall return zero. If the call to *sched_getparam*( ) is unsuccessful, the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sched_getparam*( ) function shall fail if:

[EPERM] The requesting process does not have permission to obtain the scheduling parameters of the specified process.

[ESRCH] No process can be found corresponding to that specified by *pid*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getscheduler( ), *sched_setparam*( ), *sched_setscheduler*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sched_getparam*( ) function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sched_getscheduler — get scheduling policy (**REALTIME**)

SYNOPSIS

PS

```
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

DESCRIPTION

The *sched_getscheduler*( ) function shall return the scheduling policy of the process specified by *pid*. If the value of *pid* is negative, the behavior of the *sched_getscheduler*( ) function is unspecified.

The values that can be returned by *sched_getscheduler*( ) are defined in the **<sched.h>** header.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling policy shall be returned for the process whose process ID is equal to *pid*.

If *pid* is zero, the scheduling policy shall be returned for the calling process.

RETURN VALUE

Upon successful completion, the *sched_getscheduler*( ) function shall return the scheduling policy of the specified process. If unsuccessful, the function shall return –1 and set *errno* to indicate the error.

ERRORS

The *sched_getscheduler*( ) function shall fail if:

[EPERM] The requesting process does not have permission to determine the scheduling policy of the specified process.

[ESRCH] No process can be found corresponding to that specified by *pid*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getparam( ), *sched_setparam*( ), *sched_setscheduler*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sched_getscheduler*( ) function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38567 **NAME**

38568 sched_rr_get_interval — get execution time limits (**REALTIME**)

38569 **SYNOPSIS**

38570 PS|TPS
```
#include <sched.h>

int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```
38571

38572

38573 **DESCRIPTION**

38574 The *sched_rr_get_interval*( ) function shall update the **timespec** structure referenced by the
38575 *interval* argument to contain the current execution time limit (that is, time quantum) for the
38576 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process
38577 shall be returned.

38578 **RETURN VALUE**

38579 If successful, the *sched_rr_get_interval*( ) function shall return zero. Otherwise, it shall return a
38580 value of –1 and set *errno* to indicate the error.

38581 **ERRORS**

38582 The *sched_rr_get_interval*( ) function shall fail if:

38583 [ESRCH] No process can be found corresponding to that specified by *pid*.

38584 **EXAMPLES**

38585 None.

38586 **APPLICATION USAGE**

38587 None.

38588 **RATIONALE**

38589 None.

38590 **FUTURE DIRECTIONS**

38591 None.

38592 **SEE ALSO**

38593 *sched_getparam*( ), *sched_get_priority_max*( ), *sched_getscheduler*( ), *sched_setparam*( ),
38594 *sched_setscheduler*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**

38595 **CHANGE HISTORY**

38596 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38597 **Issue 6**

38598 The *sched_rr_get_interval*( ) function is marked as part of the Process Scheduling option.

38599 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38600 implementation does not support the Process Scheduling option.

38601 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in
38602 the SYNOPSIS to PS|TPS.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38603 **NAME**

38604 sched_setparam — set scheduling parameters (**REALTIME**)

38605 **SYNOPSIS**

38606 PS
```
#include <sched.h>
```

38607
```
int sched_setparam(pid_t pid, const struct sched_param *param);
```

38608

38609 **DESCRIPTION**

38610 The *sched_setparam*( ) function shall set the scheduling parameters of the process specified by *pid*
38611 to the values specified by the **sched_param** structure pointed to by *param*. The value of the
38612 *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive
38613 priority range for the current scheduling policy of the process specified by *pid*. Higher
38614 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the
38615 behavior of the *sched_setparam*( ) function is unspecified.

38616 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
38617 parameters shall be set for the process whose process ID is equal to *pid*.

38618 If *pid* is zero, the scheduling parameters shall be set for the calling process.

38619 The conditions under which one process has permission to change the scheduling parameters of
38620 another process are implementation-defined.

38621 Implementations may require the requesting process to have the appropriate privilege to set its
38622 own scheduling parameters or those of another process.

38623 The target process, whether it is running or not running, shall be moved to the tail of the thread
38624 list for its priority.

38625 If the priority of the process specified by the *pid* argument is set higher than that of the lowest
38626 priority running process and if the specified process is ready to run, the process specified by the
38627 *pid* argument shall preempt a lowest priority running process. Similarly, if the process calling
38628 *sched_setparam*( ) sets its own priority lower than that of one or more other non-empty process
38629 lists, then the process that is the head of the highest priority list shall also preempt the calling
38630 process. Thus, in either case, the originating process might not receive notification of the
38631 completion of the requested priority change until the higher priority process has executed.

38632 SS If the scheduling policy of the target process is SCHED_SPORADIC, the value specified by the
38633 *sched_ss_low_priority* member of the *param* argument shall be any integer within the inclusive
38634 priority range for the sporadic server policy. The *sched_ss_repl_period* and *sched_ss_init_budget*
38635 members of the *param* argument shall represent the time parameters to be used by the sporadic
38636 server scheduling policy for the target process. The *sched_ss_max_repl* member of the *param*
38637 argument shall represent the maximum number of replenishments that are allowed to be
38638 pending simultaneously for the process scheduled under this scheduling policy.

38639 The specified *sched_ss_repl_period* shall be greater than or equal to the specified
38640 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

38641 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
38642 function to succeed; if not, the function shall fail.

38643 If the scheduling policy of the target process is either SCHED_FIFO or SCHED_RR, the
38644 *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init_budget* members of the *param*
38645 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process
38646 is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the effects of these members are
38647 implementation-defined; this case includes the SCHED_OTHER policy.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the current scheduling policy for the process specified by *pid* is not SCHED_FIFO, SS SCHED_RR, or SCHED_SPORADIC, the result is implementation-defined; this case includes the SCHED_OTHER policy.

The effect of this function on individual threads is dependent on the scheduling contention scope of the threads:

- For threads with system scheduling contention scope, these functions shall have no effect on their scheduling.
- For threads with process scheduling contention scope, the threads' scheduling parameters shall not be affected. However, the scheduling of these threads with respect to threads in other processes may be dependent on the scheduling parameters of their process, which are governed using these functions.

If an implementation supports a two-level scheduling model in which library threads are multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled entities for the system contention scope threads shall not be affected by these functions.

The underlying kernel-scheduled entities for the process contention scope threads shall have their scheduling parameters changed to the value specified in *param*. Kernel-scheduled entities for use by process contention scope threads that are created after this call completes shall inherit their scheduling policy and associated scheduling parameters from the process.

This function is not atomic with respect to other threads in the process. Threads may continue to execute while this function call is in the process of changing the scheduling policy for the underlying kernel-scheduled entities used by the process contention scope threads.

RETURN VALUE

If successful, the *sched_setparam*( ) function shall return zero.

If the call to *sched_setparam*( ) is unsuccessful, the priority shall remain unchanged, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sched_setparam*( ) function shall fail if:

[EINVAL] One or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified *pid*.

[EPERM] The requesting process does not have permission to set the scheduling parameters for the specified process, or does not have the appropriate privilege to invoke *sched_setparam*( ).

[ESRCH] No process can be found corresponding to that specified by *pid*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38689 **SEE ALSO**

38690 *sched_getparam*( ), *sched_getscheduler*( ), *sched_setscheduler*( ), the Base Definitions volume of
38691 IEEE Std 1003.1-2001, **<sched.h>**

38692 **CHANGE HISTORY**

38693 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38694 **Issue 6**

38695 The *sched_setparam*( ) function is marked as part of the Process Scheduling option.

38696 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38697 implementation does not support the Process Scheduling option.

38698 The following new requirements on POSIX implementations derive from alignment with the
38699 Single UNIX Specification:

38700 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
38701 added.

38702 • Sections describing two-level scheduling and atomicity of the function are added.

38703 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

38704 IEEE PASC Interpretation 1003.1 #100 is applied.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38705 **NAME**

38706 sched_setscheduler — set scheduling policy and parameters (**REALTIME**)

38707 **SYNOPSIS**

38708 PS
```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy,
    const struct sched_param *param);
```
38709
38710
38711

38712 **DESCRIPTION**

38713 The *sched_setscheduler*( ) function shall set the scheduling policy and scheduling parameters of
38714 the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure
38715 pointed to by *param*, respectively. The value of the *sched_priority* member in the **sched_param**
38716 structure shall be any integer within the inclusive priority range for the scheduling policy
38717 specified by *policy*. If the value of *pid* is negative, the behavior of the *sched_setscheduler*( )
38718 function is unspecified.

38719 The possible values for the *policy* parameter are defined in the **<sched.h>** header.

38720 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
38721 policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

38722 If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling
38723 process.

38724 The conditions under which one process has the appropriate privilege to change the scheduling
38725 parameters of another process are implementation-defined.

38726 Implementations may require that the requesting process have permission to set its own
38727 scheduling parameters or those of another process. Additionally, implementation-defined
38728 restrictions may apply as to the appropriate privileges required to set a process' own scheduling
38729 policy, or another process' scheduling policy, to a particular value.

38730 The *sched_setscheduler*( ) function shall be considered successful if it succeeds in setting the
38731 scheduling policy and scheduling parameters of the process specified by *pid* to the values
38732 specified by *policy* and the structure pointed to by *param*, respectively.

38733 SS If the scheduling policy specified by *policy* is SCHED_SPORADIC, the value specified by the
38734 *sched_ss_low_priority* member of the *param* argument shall be any integer within the inclusive
38735 priority range for the sporadic server policy. The *sched_ss_repl_period* and *sched_ss_init_budget*
38736 members of the *param* argument shall represent the time parameters used by the sporadic server
38737 scheduling policy for the target process. The *sched_ss_max_repl* member of the *param* argument
38738 shall represent the maximum number of replenishments that are allowed to be pending
38739 simultaneously for the process scheduled under this scheduling policy.

38740 The specified *sched_ss_repl_period* shall be greater than or equal to the specified
38741 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

38742 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
38743 function to succeed; if not, the function shall fail.

38744 If the scheduling policy specified by *policy* is either SCHED_FIFO or SCHED_RR, the
38745 *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init_budget* members of the *param*
38746 argument shall have no effect on the scheduling behavior.

38747 The effect of this function on individual threads is dependent on the scheduling contention
38748 scope of the threads:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38749 • For threads with system scheduling contention scope, these functions shall have no effect on
38750 their scheduling.

38751 • For threads with process scheduling contention scope, the threads' scheduling policy and
38752 associated parameters shall not be affected. However, the scheduling of these threads with
38753 respect to threads in other processes may be dependent on the scheduling parameters of their
38754 process, which are governed using these functions.

38755 If an implementation supports a two-level scheduling model in which library threads are
38756 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled
38757 entities for the system contention scope threads shall not be affected by these functions.

38758 The underlying kernel-scheduled entities for the process contention scope threads shall have
38759 their scheduling policy and associated scheduling parameters changed to the values specified in
38760 *policy* and *param*, respectively. Kernel-scheduled entities for use by process contention scope
38761 threads that are created after this call completes shall inherit their scheduling policy and
38762 associated scheduling parameters from the process.

38763 This function is not atomic with respect to other threads in the process. Threads may continue to
38764 execute while this function call is in the process of changing the scheduling policy and
38765 associated scheduling parameters for the underlying kernel-scheduled entities used by the
38766 process contention scope threads.

38767 **RETURN VALUE**

38768 Upon successful completion, the function shall return the former scheduling policy of the
38769 specified process. If the *sched_setscheduler*( ) function fails to complete successfully, the policy
38770 and scheduling parameters shall remain unchanged, and the function shall return a value of –1
38771 and set *errno* to indicate the error.

38772 **ERRORS**

38773 The *sched_setscheduler*( ) function shall fail if:

38774 [EINVAL] The value of the *policy* parameter is invalid, or one or more of the parameters
38775 contained in *param* is outside the valid range for the specified scheduling
38776 policy.

38777 [EPERM] The requesting process does not have permission to set either or both of the
38778 scheduling parameters or the scheduling policy of the specified process.

38779 [ESRCH] No process can be found corresponding to that specified by *pid*.

38780 **EXAMPLES**

38781 None.

38782 **APPLICATION USAGE**

38783 None.

38784 **RATIONALE**

38785 None.

38786 **FUTURE DIRECTIONS**

38787 None.

38788 **SEE ALSO**

38789 *sched_getparam*( ), *sched_getscheduler*( ), *sched_setparam*( ), the Base Definitions volume of
38790 IEEE Std 1003.1-2001, **<sched.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38791 **CHANGE HISTORY**

38792 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38793 **Issue 6**

38794 The *sched_setscheduler*( ) function is marked as part of the Process Scheduling option.

38795 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38796 implementation does not support the Process Scheduling option.

38797 The following new requirements on POSIX implementations derive from alignment with the
38798 Single UNIX Specification:

38799 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
38800 added.

38801 • Sections describing two-level scheduling and atomicity of the function are added.

38802 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sched_yield — yield the processor

SYNOPSIS

PS|THR

```
#include <sched.h>

int sched_yield(void);
```

DESCRIPTION

The *sched_yield*( ) function shall force the running thread to relinquish the processor until it again becomes the head of its thread list. It takes no arguments.

RETURN VALUE

The *sched_yield*( ) function shall return 0 if it completes successfully; otherwise, it shall return a value of –1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The *sched_yield*( ) function is now marked as part of the Process Scheduling and Threads options.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38832 **NAME**

38833 seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

38834 **SYNOPSIS**

38835 XSI
```
#include <stdlib.h>
```

38836
```
unsigned short *seed48(unsigned short seed16v[3]);
```

38837

38838 **DESCRIPTION**

38839 Refer to *drand48*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

seekdir — set the position of a directory stream

SYNOPSIS

XSI
```
#include <dirent.h>

void seekdir(DIR *dirp, long loc);
```

DESCRIPTION

The *seekdir*( ) function shall set the position of the next *readdir*( ) operation on the directory stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been returned from an earlier call to *telldir*( ). The new position reverts to the one associated with the directory stream when *telldir*( ) was performed.

If the value of *loc* was not obtained from an earlier call to *telldir*( ), or if a call to *rewinddir*( ) occurred between the call to *telldir*( ) and the call to *seekdir*( ), the results of subsequent calls to *readdir*( ) are unspecified.

RETURN VALUE

The *seekdir*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The original standard developers perceived that there were restrictions on the use of the *seekdir*( ) and *telldir*( ) functions related to implementation details, and for that reason these functions need not be supported on all POSIX-conforming systems. They are required on implementations supporting the XSI extension.

One of the perceived problems of implementation is that returning to a given point in a directory is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-trees, hashing functions, or other similar mechanisms to order their directories are considered. The definition of *seekdir*( ) and *telldir*( ) does not specify whether, when using these interfaces, a given directory entry will be seen at all, or more than once.

On systems not supporting these functions, their capability can sometimes be accomplished by saving a filename found by *readdir*( ) and later using *rewinddir*( ) and a loop on *readdir*( ) to relocate the position from which the filename was saved.

FUTURE DIRECTIONS

None.

SEE ALSO

opendir( ), *readdir*( ), *telldir*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**, **<stdio.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 2.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

38882 **Issue 6**

38883 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38884 **NAME**

38885 select — synchronous I/O multiplexing

38886 **SYNOPSIS**

38887 `#include <sys/time.h>`

38888 `int select(int` *`nfds`*`, fd_set *restrict` *`readfds`*`,`

38889 `fd_set *restrict` *`writefds`*`, fd_set *restrict` *`errorfds`*`,`

38890 `struct timeval *restrict` *`timeout`*`);`

38891

38892 **DESCRIPTION**

38893 Refer to *pselect*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

38894 **NAME**

38895 sem_close — close a named semaphore (**REALTIME**)

38896 **SYNOPSIS**

38897 SEM
```
#include <semaphore.h>

int sem_close(sem_t *sem);
```
38898

38899

38900 **DESCRIPTION**

38901 The *sem_close*( ) function shall indicate that the calling process is finished using the named
38902 semaphore indicated by *sem*. The effects of calling *sem_close*( ) for an unnamed semaphore (one
38903 created by *sem_init*( )) are undefined. The *sem_close*( ) function shall deallocate (that is, make
38904 available for reuse by a subsequent *sem_open*( ) by this process) any system resources allocated
38905 by the system for use by this process for this semaphore. The effect of subsequent use of the
38906 semaphore indicated by *sem* by this process is undefined. If the semaphore has not been
38907 removed with a successful call to *sem_unlink*( ), then *sem_close*( ) has no effect on the state of the
38908 semaphore. If the *sem_unlink*( ) function has been successfully invoked for *name* after the most
38909 recent call to *sem_open*( ) with O_CREAT for this semaphore, then when all processes that have
38910 opened the semaphore close it, the semaphore is no longer accessible.

38911 **RETURN VALUE**

38912 Upon successful completion, a value of zero shall be returned. Otherwise, a value of –1 shall be
38913 returned and *errno* set to indicate the error.

38914 **ERRORS**

38915 The *sem_close*( ) function shall fail if:

38916 [EINVAL] The *sem* argument is not a valid semaphore descriptor.

38917 **EXAMPLES**

38918 None.

38919 **APPLICATION USAGE**

38920 The *sem_close*( ) function is part of the Semaphores option and need not be available on all
38921 implementations.

38922 **RATIONALE**

38923 None.

38924 **FUTURE DIRECTIONS**

38925 None.

38926 **SEE ALSO**

38927 *semctl*( ), *semget*( ), *semop*( ), *sem_init*( ), *sem_open*( ), *sem_unlink*( ), the Base Definitions volume of
38928 IEEE Std 1003.1-2001, **<semaphore.h>**

38929 **CHANGE HISTORY**

38930 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38931 **Issue 6**

38932 The *sem_close*( ) function is marked as part of the Semaphores option.

38933 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38934 implementation does not support the Semaphores option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sem_destroy — destroy an unnamed semaphore (**REALTIME**)

SYNOPSIS

SEM
```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
```

DESCRIPTION

The *sem_destroy*( ) function shall destroy the unnamed semaphore indicated by *sem*. Only a semaphore that was created using *sem_init*( ) may be destroyed using *sem_destroy*( ); the effect of calling *sem_destroy*( ) with a named semaphore is undefined. The effect of subsequent use of the semaphore *sem* is undefined until *sem* is reinitialized by another call to *sem_init*( ).

It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The effect of destroying a semaphore upon which other threads are currently blocked is undefined.

RETURN VALUE

Upon successful completion, a value of zero shall be returned. Otherwise, a value of –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sem_destroy*( ) function shall fail if:

[EINVAL] The *sem* argument is not a valid semaphore.

The *sem_destroy*( ) function may fail if:

[EBUSY] There are currently processes blocked on the semaphore.

EXAMPLES

None.

APPLICATION USAGE

The *sem_destroy*( ) function is part of the Semaphores option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_init*( ), *sem_open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_destroy*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sem_getvalue — get the value of a semaphore (**REALTIME**)

SYNOPSIS

SEM
```
#include <semaphore.h>

int sem_getvalue(sem_t *restrict sem, int *restrict sval);
```

DESCRIPTION

The *sem_getvalue*( ) function shall update the location referenced by the *sval* argument to have the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The updated value represents an actual semaphore value that occurred at some unspecified time during the call, but it need not be the actual value of the semaphore when it is returned to the calling process.

If *sem* is locked, then the object to which *sval* points shall either be set to zero or to a negative number whose absolute value represents the number of processes waiting for the semaphore at some unspecified time during the call.

RETURN VALUE

Upon successful completion, the *sem_getvalue*( ) function shall return a value of zero. Otherwise, it shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sem_getvalue*( ) function shall fail if:

[EINVAL] The *sem* argument does not refer to a valid semaphore.

EXAMPLES

None.

APPLICATION USAGE

The *sem_getvalue*( ) function is part of the Semaphores option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_getvalue*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *sem_getvalue*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

39017 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied. |

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

39018 **NAME**

39019 sem_init — initialize an unnamed semaphore (**REALTIME**)

39020 **SYNOPSIS**

39021 SEM
```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned value);
```
39022

39023

39024 **DESCRIPTION**

39025 The *sem_init*( ) function shall initialize the unnamed semaphore referred to by *sem*. The value of
39026 the initialized semaphore shall be *value*. Following a successful call to *sem_init*( ), the semaphore
39027 may be used in subsequent calls to *sem_wait*( ), *sem_trywait*( ), *sem_post*( ), and *sem_destroy*( ).
39028 This semaphore shall remain usable until the semaphore is destroyed.

39029 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;
39030 in this case, any process that can access the semaphore *sem* can use *sem* for performing
39031 *sem_wait*( ), *sem_trywait*( ), *sem_post*( ), and *sem_destroy*( ) operations.

39032 Only *sem* itself may be used for performing synchronization. The result of referring to copies of
39033 *sem* in calls to *sem_wait*( ), *sem_trywait*( ), *sem_post*( ), and *sem_destroy*( ) is undefined.

39034 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any
39035 thread in this process can use *sem* for performing *sem_wait*( ), *sem_trywait*( ), *sem_post*( ), and
39036 *sem_destroy*( ) operations. The use of the semaphore by threads other than those created in the
39037 same process is undefined.

39038 Attempting to initialize an already initialized semaphore results in undefined behavior.

39039 **RETURN VALUE**

39040 Upon successful completion, the *sem_init*( ) function shall initialize the semaphore in *sem*.
39041 Otherwise, it shall return –1 and set *errno* to indicate the error.

39042 **ERRORS**

39043 The *sem_init*( ) function shall fail if:

39044 [EINVAL] The *value* argument exceeds {SEM_VALUE_MAX}.

39045 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the
39046 limit on semaphores ({SEM_NSEMS_MAX}) has been reached.

39047 [EPERM] The process lacks the appropriate privileges to initialize the semaphore.

39048 **EXAMPLES**

39049 None.

39050 **APPLICATION USAGE**

39051 The *sem_init*( ) function is part of the Semaphores option and need not be available on all
39052 implementations.

39053 **RATIONALE**

39054 Although this volume of IEEE Std 1003.1-2001 fails to specify a successful return value, it is
39055 likely that a later version may require the implementation to return a value of zero if the call to
39056 *sem_init*( ) is successful.

39057 **FUTURE DIRECTIONS**

39058 None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

39059 **SEE ALSO**

39060 *sem_destroy*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_wait*( ), the Base Definitions
39061 volume of IEEE Std 1003.1-2001, **<semaphore.h>**

39062 **CHANGE HISTORY**

39063 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39064 **Issue 6**

39065 The *sem_init*( ) function is marked as part of the Semaphores option.

39066 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39067 implementation does not support the Semaphores option.

39068 The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with
39069 IEEE Std 1003.1d-1999.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sem_open — initialize and open a named semaphore (**REALTIME**)

SYNOPSIS

SEM
```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, ...);
```

DESCRIPTION

The *sem_open*( ) function shall establish a connection between a named semaphore and a process. Following a call to *sem_open*( ) with semaphore name *name*, the process may reference the semaphore associated with *name* using the address returned from the call. This semaphore may be used in subsequent calls to *sem_wait*( ), *sem_trywait*( ), *sem_post*( ), and *sem_close*( ). The semaphore remains usable by this process until the semaphore is closed by a successful call to *sem_close*( ), *_exit*( ), or one of the *exec* functions.

The *oflag* argument controls whether the semaphore is created or merely accessed by the call to *sem_open*( ). The following flag bits may be set in *oflag*:

O_CREAT This flag is used to create a semaphore if it does not already exist. If O_CREAT is set and the semaphore already exists, then O_CREAT has no effect, except as noted under O_EXCL. Otherwise, *sem_open*( ) creates a named semaphore. The O_CREAT flag requires a third and a fourth argument: *mode*, which is of type **mode_t**, and *value*, which is of type **unsigned**. The semaphore is created with an initial value of *value*. Valid initial values for semaphores are less than or equal to {SEM_VALUE_MAX}.

The user ID of the semaphore is set to the effective user ID of the process; the group ID of the semaphore is set to a system default group ID or to the effective group ID of the process. The permission bits of the semaphore are set to the value of the *mode* argument except those set in the file mode creation mask of the process. When bits in *mode* other than the file permission bits are specified, the effect is unspecified.

After the semaphore named *name* has been created by *sem_open*( ) with the O_CREAT flag, other processes can connect to the semaphore by calling *sem_open*( ) with the same value of *name*.

O_EXCL If O_EXCL and O_CREAT are set, *sem_open*( ) fails if the semaphore *name* exists. The check for the existence of the semaphore and the creation of the semaphore if it does not exist are atomic with respect to other processes executing *sem_open*( ) with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the effect is undefined.

If flags other than O_CREAT and O_EXCL are specified in the *oflag* parameter, the effect is unspecified.

The *name* argument points to a string naming a semaphore object. It is unspecified whether the name appears in the file system and is visible to functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname. If *name* begins with the slash character, then processes calling *sem_open*( ) with the same value of *name* shall refer to the same semaphore object, as long as that name has not been removed. If *name* does not begin with the slash character, the effect is implementation-defined. The interpretation of slash characters other than the leading slash character in *name* is implementation-defined.

If a process makes multiple successful calls to *sem_open*( ) with the same value for *name*, the same semaphore address shall be returned for each such successful call, provided that there

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

have been no calls to *sem_unlink*( ) for this semaphore.

References to copies of the semaphore produce undefined results.

RETURN VALUE

Upon successful completion, the *sem_open*( ) function shall return the address of the semaphore. Otherwise, it shall return a value of SEM_FAILED and set *errno* to indicate the error. The symbol SEM_FAILED is defined in the **<semaphore.h>** header. No successful return from *sem_open*( ) shall return the value SEM_FAILED.

ERRORS

If any of the following conditions occur, the *sem_open*( ) function shall return SEM_FAILED and set *errno* to the corresponding value:

[EACCES] The named semaphore exists and the permissions specified by *oflag* are denied, or the named semaphore does not exist and permission to create the named semaphore is denied.

[EEXIST] O_CREAT and O_EXCL are set and the named semaphore already exists.

[EINTR] The *sem_open*( ) operation was interrupted by a signal.

[EINVAL] The *sem_open*( ) operation is not supported for the given name, or O_CREAT was specified in *oflag* and *value* was greater than {SEM_VALUE_MAX}.

[EMFILE] Too many semaphore descriptors or file descriptors are currently in use by this process.

[ENAMETOOLONG] The length of the *name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENFILE] Too many semaphores are currently open in the system.

[ENOENT] O_CREAT is not set and the named semaphore does not exist.

[ENOSPC] There is insufficient space for the creation of the new named semaphore.

EXAMPLES

None.

APPLICATION USAGE

The *sem_open*( ) function is part of the Semaphores option and need not be available on all implementations.

RATIONALE

Early drafts required an error return value of –1 with the type **sem_t *** for the *sem_open*( ) function, which is not guaranteed to be portable across implementations. The revised text provides the symbolic error code SEM_FAILED to eliminate the type conflict.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_close*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_unlink*( ), *sem_wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 6

The *sem_open*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

sem_post — unlock a semaphore (**REALTIME**)

SYNOPSIS

SEM

```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

DESCRIPTION

The *sem_post*( ) function shall unlock the semaphore referenced by *sem* by performing a semaphore unlock operation on that semaphore.

If the semaphore value resulting from this operation is positive, then no threads were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented.

If the value of the semaphore resulting from this operation is zero, then one of the threads blocked waiting for the semaphore shall be allowed to return successfully from its call to *sem_wait*( ). PS If the Process Scheduling option is supported, the thread to be unblocked shall be chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked threads. In the case of the schedulers SCHED_FIFO and SCHED_RR, the highest priority waiting thread shall be unblocked, and if there is more than one highest priority thread blocked waiting for the semaphore, then the highest priority thread that has been waiting the longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread to unblock is unspecified.

SS If the Process Sporadic Server option is supported, and the scheduling policy is SCHED_SPORADIC, the semantics are as per SCHED_FIFO above.

The *sem_post*( ) function shall be reentrant with respect to signals and may be invoked from a signal-catching function.

RETURN VALUE

If successful, the *sem_post*( ) function shall return zero; otherwise, the function shall return –1 and set *errno* to indicate the error.

ERRORS

The *sem_post*( ) function shall fail if:

[EINVAL] The *sem* argument does not refer to a valid semaphore.

EXAMPLES

None.

APPLICATION USAGE

The *sem_post*( ) function is part of the Semaphores option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

39206 **CHANGE HISTORY**

39207 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39208 **Issue 6**

39209 The *sem_post*( ) function is marked as part of the Semaphores option.

39210 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39211 implementation does not support the Semaphores option.

39212 The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with
39213 IEEE Std 1003.1d-1999.

39214 SCHED_SPORADIC is added to the list of scheduling policies for which the thread that is to be
39215 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sem_timedwait — lock a semaphore (**ADVANCED REALTIME**)

SYNOPSIS

SEM TMO
```
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
    const struct timespec *restrict abs_timeout);
```

DESCRIPTION

The *sem_timedwait*( ) function shall lock the semaphore referenced by *sem* as in the *sem_wait*( ) function. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore by performing a *sem_post*( ) function, this wait shall be terminated when the specified timeout expires.

The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout* has already been passed at the time of the call.

TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock. If the Timers option is not supported, the timeout shall be based on the system clock as returned by the *time*( ) function. The resolution of the timeout shall be the resolution of the clock on which it is based. The **timespec** data type is defined as a structure in the **<time.h>** header.

Under no circumstance shall the function fail with a timeout if the semaphore can be locked immediately. The validity of the *abs_timeout* need not be checked if the semaphore can be locked immediately.

RETURN VALUE

The *sem_timedwait*( ) function shall return zero if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sem_timedwait*( ) function shall fail if:

[EINVAL] The *sem* argument does not refer to a valid semaphore.

[EINVAL] The process or thread would have blocked, and the *abs_timeout* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

[ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

The *sem_timedwait*( ) function may fail if:

[EDEADLK] A deadlock condition was detected.

[EINTR] A signal interrupted this function.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

The *sem_timedwait*( ) function is part of the Semaphores and Timeouts options and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sem_post( ), *sem_trywait*( ), *sem_wait*( ), *semctl*( ), *semget*( ), *semop*( ), *time*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sem_trywait, sem_wait — lock a semaphore (**REALTIME**)

SYNOPSIS

SEM

```
#include <semaphore.h>

int sem_trywait(sem_t *sem);
int sem_wait(sem_t *sem);
```

DESCRIPTION

The *sem_trywait*( ) function shall lock the semaphore referenced by *sem* only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not lock the semaphore.

The *sem_wait*( ) function shall lock the semaphore referenced by *sem* by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread shall not return from the call to *sem_wait*( ) until it either locks the semaphore or the call is interrupted by a signal.

Upon successful return, the state of the semaphore shall be locked and shall remain locked until the *sem_post*( ) function is executed and returns successfully.

The *sem_wait*( ) function is interruptible by the delivery of a signal.

RETURN VALUE

The *sem_trywait*( ) and *sem_wait*( ) functions shall return zero if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sem_trywait*( ) and *sem_wait*( ) functions shall fail if:

[EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the *sem_trywait*( ) operation (*sem_trywait*( ) only).

[EINVAL] The *sem* argument does not refer to a valid semaphore.

The *sem_trywait*( ) and *sem_wait*( ) functions may fail if:

[EDEADLK] A deadlock condition was detected.

[EINTR] A signal interrupted this function.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

The *sem_trywait*( ) and *sem_wait*( ) functions are part of the Semaphores option and need not be provided on all implementations.

RATIONALE

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_post*( ), *sem_timedwait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_trywait*( ) and *sem_wait*( ) functions are marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait*( ) function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sem_unlink — remove a named semaphore (**REALTIME**)

SYNOPSIS

SEM

```
#include <semaphore.h>

int sem_unlink(const char *name);
```

DESCRIPTION

The *sem_unlink*( ) function shall remove the semaphore named by the string *name*. If the semaphore named by *name* is currently referenced by other processes, then *sem_unlink*( ) shall have no effect on the state of the semaphore. If one or more processes have the semaphore open when *sem_unlink*( ) is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by calls to *sem_close*( ), *_exit*( ), or *exec*. Calls to *sem_open*( ) to recreate or reconnect to the semaphore refer to a new semaphore after *sem_unlink*( ) is called. The *sem_unlink*( ) call shall not block until all references have been destroyed; it shall return immediately.

RETURN VALUE

Upon successful completion, the *sem_unlink*( ) function shall return a value of 0. Otherwise, the semaphore shall not be changed and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sem_unlink*( ) function shall fail if:

[EACCES] Permission is denied to unlink the named semaphore.

[ENAMETOOLONG]
The length of the *name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] The named semaphore does not exist.

EXAMPLES

None.

APPLICATION USAGE

The *sem_unlink*( ) function is part of the Semaphores option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_close*( ), *sem_open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_unlink*( ) function is marked as part of the Semaphores option.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

39366 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39367 implementation does not support the Semaphores option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sem_wait — lock a semaphore (**REALTIME**)

SYNOPSIS

SEM

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
```

DESCRIPTION

Refer to *sem_trywait*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

semctl — XSI semaphore control operations

SYNOPSIS

XSI
```
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

DESCRIPTION

The *semctl*( ) function operates on XSI semaphores (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *semctl*( ) function provides a variety of semaphore control operations as specified by *cmd*. The fourth argument is optional and depends upon the operation requested. If required, it is of type **union semun**, which the application shall explicitly declare:

```
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short  *array;
} arg;
```

The following semaphore control operations as specified by *cmd* are executed with respect to the semaphore specified by *semid* and *semnum*. The level of permission required for each operation is shown with each command; see Section 2.7 (on page 39). The symbolic names for the values of *cmd* are defined in the **<sys/sem.h>** header:

GETVAL — Return the value of *semval*; see **<sys/sem.h>**. Requires read permission.

SETVAL — Set the value of *semval* to *arg.val*, where *arg* is the value of the fourth argument to *semctl*( ). When this command is successfully executed, the *semadj* value corresponding to the specified semaphore in all processes is cleared. Requires alter permission; see Section 2.7 (on page 39).

GETPID — Return the value of *sempid*. Requires read permission.

GETNCNT — Return the value of *semncnt*. Requires read permission.

GETZCNT — Return the value of *semzcnt*. Requires read permission.

The following values of *cmd* operate on each *semval* in the set of semaphores:

GETALL — Return the value of *semval* for each semaphore in the semaphore set and place into the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl*( ). Requires read permission.

SETALL — Set the value of *semval* for each semaphore in the semaphore set according to the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl*( ). When this command is successfully executed, the *semadj* values corresponding to each specified semaphore in all processes are cleared. Requires alter permission.

The following values of *cmd* are also available:

IPC_STAT — Place the current value of each member of the **semid_ds** data structure associated with *semid* into the structure pointed to by *arg.buf*, where *arg* is the fourth argument to *semctl*( ). The contents of this structure are defined in

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

<sys/sem.h>. Requires read permission.

IPC_SET — Set the value of the following members of the **semid_ds** data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*, where *arg* is the fourth argument to *semctl*( ):

```
sem_perm.uid
sem_perm.gid
sem_perm.mode
```

The mode bits specified in Section 2.7.1 (on page 40) are copied into the corresponding bits of the *sem_perm.mode* associated with *semid*. The stored values of any other bits are unspecified.

This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *sem_perm.cuid* or *sem_perm.uid* in the **semid_ds** data structure associated with *semid*.

IPC_RMID — Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and **semid_ds** data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *sem_perm.cuid* or *sem_perm.uid* in the **semid_ds** data structure associated with *semid*.

RETURN VALUE

If successful, the value returned by *semctl*( ) depends on *cmd* as follows:

GETVAL	The value of *semval*.
GETPID	The value of *sempid*.
GETNCNT	The value of *semncnt*.
GETZCNT	The value of *semzcnt*.
All others	0.

Otherwise, *semctl*( ) shall return –1 and set *errno* to indicate the error.

ERRORS

The *semctl*( ) function shall fail if:

[EACCES] — Operation permission is denied to the calling process; see Section 2.7 (on page 39).

[EINVAL] — The value of *semid* is not a valid semaphore identifier, or the value of *semnum* is less than 0 or greater than or equal to *sem_nsems*, or the value of *cmd* is not a valid command.

[EPERM] — The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of *sem_perm.cuid* or *sem_perm.uid* in the data structure associated with *semid*.

[ERANGE] — The argument *cmd* is equal to SETVAL or SETALL and the value to which *semval* is to be set is greater than the system-imposed maximum.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The fourth parameter in the SYNOPSIS section is now specified as "..." in order to avoid a clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for backwards-compatibility.

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *semget*( ), *semop*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_unlink*( ), *sem_wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/sem.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to the APPLICATION USAGE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

semget — get set of XSI semaphores

SYNOPSIS

XSI
```
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

DESCRIPTION

The *semget*( ) function operates on XSI semaphores (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *semget*( ) function shall return the semaphore identifier associated with *key*.

A semaphore identifier with its associated **semid_ds** data structure and its associated set of *nsems* semaphores (see **<sys/sem.h>**) is created for *key* if one of the following is true:

- The argument *key* is equal to IPC_PRIVATE.
- The argument *key* does not already have a semaphore identifier associated with it and (*semflg* &IPC_CREAT) is non-zero.

Upon creation, the **semid_ds** data structure associated with the new semaphore identifier is initialized as follows:

- In the operation permissions structure *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid*, and *sem_perm.gid* shall be set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of *sem_perm.mode* shall be set equal to the low-order 9 bits of *semflg*.
- The variable *sem_nsems* shall be set equal to the value of *nsems*.
- The variable *sem_otime* shall be set equal to 0 and *sem_ctime* shall be set equal to the current time.
- The data structure associated with each semaphore in the set shall not be initialized. The *semctl*( ) function with the command SETVAL or SETALL can be used to initialize each semaphore.

RETURN VALUE

Upon successful completion, *semget*( ) shall return a non-negative integer, namely a semaphore identifier; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *semget*( ) function shall fail if:

[EACCES] A semaphore identifier exists for *key*, but operation permission as specified by the low-order 9 bits of *semflg* would not be granted; see Section 2.7 (on page 39).

[EEXIST] A semaphore identifier exists for the argument *key* but ((*semflg* &IPC_CREAT) &&(*semflg* &IPC_EXCL)) is non-zero.

[EINVAL] The value of *nsems* is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument *key*, but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to 0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

39527 [ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg*
39528 &IPC_CREAT) is equal to 0.

39529 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the
39530 maximum number of allowed semaphores system-wide would be exceeded.

39531 **EXAMPLES**

39532 **Creating a Semaphore Identifier**

39533 The following example gets a unique semaphore key using the *ftok*( ) function, then gets a
39534 semaphore ID associated with that key using the *semget*( ) function (the first call also tests to
39535 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
39536 shown by the second call to *semget*( ). In creating the semaphore for the queuing process, the
39537 program attempts to create one semaphore with read/write permission for all. It also uses the
39538 IPC_EXCL flag, which forces *semget*( ) to fail if the semaphore already exists.

39539 After creating the semaphore, the program uses a call to *semop*( ) to initialize it to the values in
39540 the *sbuf* array. The number of processes that can execute concurrently without queuing is
39541 initially set to 2. The final call to *semget*( ) creates a semaphore identifier that can be used later in
39542 the program.

```
39543 #include <sys/types.h>
39544 #include <stdio.h>
39545 #include <sys/ipc.h>
39546 #include <sys/sem.h>
39547 #include <sys/stat.h>
39548 #include <errno.h>
39549 #include <unistd.h>
39550 #include <stdlib.h>
39551 #include <pwd.h>
39552 #include <fcntl.h>
39553 #include <limits.h>
39554 ...
39555 key_t semkey;
39556 int semid, pfd, fv;
39557 struct sembuf sbuf;
39558 char *lgn;
39559 char filename[PATH_MAX+1];
39560 struct stat outstat;
39561 struct passwd *pw;
39562 ...
39563 /* Get unique key for semaphore. */
39564 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39565     perror("IPC error: ftok"); exit(1);
39566 }
39567 /* Get semaphore ID associated with this key. */
39568 if ((semid = semget(semkey, 0, 0)) == -1) {
39569     /* Semaphore does not exist - Create. */
39570     if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39571         S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39572     {
39573         /* Initialize the semaphore. */
39574         sbuf.sem_num = 0;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                sbuf.sem_op = 2;  /* This is the number of runs
                                     without queuing. */
                sbuf.sem_flg = 0;
                if (semop(semid, &sbuf, 1) == -1) {
                    perror("IPC error: semop"); exit(1);
                }
            }
            else if (errno == EEXIST) {
                if ((semid = semget(semkey, 0, 0)) == -1) {
                    perror("IPC error 1: semget"); exit(1);
                }
            }
            else {
                perror("IPC error 2: semget"); exit(1);
            }
        }
        ...
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *semctl*( ), *semop*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_unlink*( ), *sem_wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/sem.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

semop — XSI semaphore operations

SYNOPSIS

XSI
```
#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, size_t nsops);
```

DESCRIPTION

The *semop*( ) function operates on XSI semaphores (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *semop*( ) function shall perform atomically a user-defined array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by the argument *semid*.

The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The implementation shall not modify elements of this array unless the application uses implementation-defined extensions.

The argument *nsops* is the number of such structures in the array.

Each structure, **sembuf**, includes the following members:

Member Type	Member Name	Description
short	*sem_num*	Semaphore number.
short	*sem_op*	Semaphore operation.
short	*sem_flg*	Operation flags.

Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore specified by *semid* and *sem_num*.

The variable *sem_op* specifies one of three semaphore operations:

1. If *sem_op* is a negative integer and the calling process has alter permission, one of the following shall occur:
   - If *semval*(see **<sys/sem.h>**) is greater than or equal to the absolute value of *sem_op*, the absolute value of *sem_op* is subtracted from *semval*. Also, if (*sem_flg* &SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the calling process' *semadj* value for the specified semaphore.
   - If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is non-zero, *semop*( ) shall return immediately.
   - If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is 0, *semop*( ) shall increment the *semncnt* associated with the specified semaphore and suspend execution of the calling thread until one of the following conditions occurs:
     — The value of *semval* becomes greater than or equal to the absolute value of *sem_op*. When this occurs, the value of *semncnt* associated with the specified semaphore shall be decremented, the absolute value of *sem_op* shall be subtracted from *semval* and, if (*sem_flg* &SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the calling process' *semadj* value for the specified semaphore.
     — The *semid* for which the calling thread is awaiting action is removed from the system. When this occurs, *errno* shall be set equal to [EIDRM] and –1 shall be

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

39654 returned.

39655 — The calling thread receives a signal that is to be caught. When this occurs, the value
39656 of *semncnt* associated with the specified semaphore shall be decremented, and the
39657 calling thread shall resume execution in the manner prescribed in *sigaction*( ).

39658 2. If *sem_op* is a positive integer and the calling process has alter permission, the value of
39659 *sem_op* shall be added to *semval* and, if (*sem_flg* &SEM_UNDO) is non-zero, the value of
39660 *sem_op* shall be subtracted from the calling process' *semadj* value for the specified
39661 semaphore.

39662 3. If *sem_op* is 0 and the calling process has read permission, one of the following shall occur:

39663 • If *semval* is 0, *semop*( ) shall return immediately.

39664 • If *semval* is non-zero and (*sem_flg* &IPC_NOWAIT) is non-zero, *semop*( ) shall return
39665 immediately.

39666 • If *semval* is non-zero and (*sem_flg* &IPC_NOWAIT) is 0, *semop*( ) shall increment the
39667 *semzcnt* associated with the specified semaphore and suspend execution of the calling
39668 thread until one of the following occurs:

39669 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated with
39670 the specified semaphore shall be decremented.

39671 — The *semid* for which the calling thread is awaiting action is removed from the
39672 system. When this occurs, *errno* shall be set equal to [EIDRM] and –1 shall be
39673 returned.

39674 — The calling thread receives a signal that is to be caught. When this occurs, the value
39675 of *semzcnt* associated with the specified semaphore shall be decremented, and the
39676 calling thread shall resume execution in the manner prescribed in *sigaction*( ).

39677 Upon successful completion, the value of *sempid* for each semaphore specified in the array
39678 pointed to by *sops* shall be set equal to the process ID of the calling process.

39679 **RETURN VALUE**

39680 Upon successful completion, *semop*( ) shall return 0; otherwise, it shall return –1 and set *errno* to
39681 indicate the error.

39682 **ERRORS**

39683 The *semop*( ) function shall fail if:

39684	[E2BIG]	The value of *nsops* is greater than the system-imposed maximum.
39685 39686	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 39).
39687 39688	[EAGAIN]	The operation would result in suspension of the calling process but (*sem_flg* &IPC_NOWAIT) is non-zero.
39689 39690	[EFBIG]	The value of *sem_num* is less than 0 or greater than or equal to the number of semaphores in the set associated with *semid*.
39691	[EIDRM]	The semaphore identifier *semid* is removed from the system.
39692	[EINTR]	The *semop*( ) function was interrupted by a signal.
39693 39694 39695	[EINVAL]	The value of *semid* is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[ENOSPC] The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.

[ERANGE] An operation would cause a *semval* to overflow the system-imposed limit, or an operation would cause a *semadj* value to overflow the system-imposed limit.

EXAMPLES

Setting Values in Semaphores

The following example sets the values of the two semaphores associated with the *semid* identifier to the values contained in the *sb* array.

```
#include <sys/sem.h>
...
int semid;
struct sembuf sb[2];
int nsops = 2;
int result;

/* Adjust value of semaphore in the semaphore array semid. */
sb[0].sem_num = 0;
sb[0].sem_op = -1;
sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
sb[1].sem_num = 1;
sb[1].sem_op =  1;
sb[1].sem_flg = 0;

result = semop(semid, sb, nsops);
```

Creating a Semaphore Identifier

The following example gets a unique semaphore key using the *ftok*( ) function, then gets a semaphore ID associated with that key using the *semget*( ) function (the first call also tests to make sure the semaphore exists). If the semaphore does not exist, the program creates it, as shown by the second call to *semget*( ). In creating the semaphore for the queuing process, the program attempts to create one semaphore with read/write permission for all. It also uses the IPC_EXCL flag, which forces *semget*( ) to fail if the semaphore already exists.

After creating the semaphore, the program uses a call to *semop*( ) to initialize it to the values in the *sbuf* array. The number of processes that can execute concurrently without queuing is initially set to 2. The final call to *semget*( ) creates a semaphore identifier that can be used later in the program.

The final call to *semop*( ) acquires the semaphore and waits until it is free; the SEM_UNDO option releases the semaphore when the process exits, waiting until there are less than two processes running concurrently.

```
#include <sys/types.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        #include <pwd.h>
        #include <fcntl.h>
        #include <limits.h>
        ...
        key_t semkey;
        int semid, pfd, fv;
        struct sembuf sbuf;
        char *lgn;
        char filename[PATH_MAX+1];
        struct stat outstat;
        struct passwd *pw;
        ...
        /* Get unique key for semaphore. */
        if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
            perror("IPC error: ftok"); exit(1);
        }
        /* Get semaphore ID associated with this key. */
        if ((semid = semget(semkey, 0, 0)) == -1) {
            /* Semaphore does not exist - Create. */
            if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
                S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
            {
                /* Initialize the semaphore. */
                sbuf.sem_num = 0;
                sbuf.sem_op = 2;  /* This is the number of runs without queuing. */
                sbuf.sem_flg = 0;
                if (semop(semid, &sbuf, 1) == -1) {
                    perror("IPC error: semop"); exit(1);
                }
            }
            else if (errno == EEXIST) {
                if ((semid = semget(semkey, 0, 0)) == -1) {
                    perror("IPC error 1: semget"); exit(1);
                }
            }
            else {
                perror("IPC error 2: semget"); exit(1);
            }
        }
        ...
        sbuf.sem_num = 0;
        sbuf.sem_op = -1;
        sbuf.sem_flg = SEM_UNDO;
        if (semop(semid, &sbuf, 1) == -1) {
            perror("IPC Error: semop"); exit(1);
        }
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

39792 **RATIONALE**

39793 None.

39794 **FUTURE DIRECTIONS**

39795 None.

39796 **SEE ALSO**

39797 Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit*( ), *fork*( ), *semctl*( ), *semget*( ),
39798 *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_unlink*( ),
39799 *sem_wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/ipc.h>**, **<sys/sem.h>**,
39800 **<sys/types.h>**

39801 **CHANGE HISTORY**

39802 First released in Issue 2. Derived from Issue 2 of the SVID.

39803 **Issue 5**

39804 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
39805 DIRECTIONS to a new APPLICATION USAGE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

39806 **NAME**

39807 send — send a message on a socket

39808 **SYNOPSIS**

39809 `#include <sys/socket.h>`

39810 `ssize_t send(int socket, const void *buffer, size_t length, int flags);`

39811 **DESCRIPTION**

39812 The *send*( ) function shall initiate transmission of a message from the specified socket to its peer.
39813 The *send*( ) function shall send a message only when the socket is connected (including when the
39814 peer of a connectionless socket has been set via *connect*( )).

39815 The *send*( ) function takes the following arguments:

39816 *socket* Specifies the socket file descriptor.

39817 *buffer* Points to the buffer containing the message to send.

39818 *length* Specifies the length of the message in bytes.

39819 *flags* Specifies the type of message transmission. Values of this argument are
39820 formed by logically OR'ing zero or more of the following flags:

39821 MSG_EOR Terminates a record (if supported by the protocol).

39822 MSG_OOB Sends out-of-band data on sockets that support out-of-band
39823 communications. The significance and semantics of out-of-
39824 band data are protocol-specific.

39825 The length of the message to be sent is specified by the *length* argument. If the message is too
39826 long to pass through the underlying protocol, *send*( ) shall fail and no data shall be transmitted.

39827 Successful completion of a call to *send*( ) does not guarantee delivery of the message. A return
39828 value of –1 indicates only locally-detected errors.

39829 If space is not available at the sending socket to hold the message to be transmitted, and the
39830 socket file descriptor does not have O_NONBLOCK set, *send*( ) shall block until space is
39831 available. If space is not available at the sending socket to hold the message to be transmitted,
39832 and the socket file descriptor does have O_NONBLOCK set, *send*( ) shall fail. The *select*( ) and
39833 *poll*( ) functions can be used to determine when it is possible to send more data.

39834 The socket in use may require the process to have appropriate privileges to use the *send*( )
39835 function.

39836 **RETURN VALUE**

39837 Upon successful completion, *send*( ) shall return the number of bytes sent. Otherwise, –1 shall be
39838 returned and *errno* set to indicate the error.

39839 **ERRORS**

39840 The *send*( ) function shall fail if:

39841 [EAGAIN] or [EWOULDBLOCK]
39842 The socket's file descriptor is marked O_NONBLOCK and the requested
39843 operation would block.

39844 [EBADF] The *socket* argument is not a valid file descriptor.

39845 [ECONNRESET] A connection was forcibly closed by a peer.

39846 [EDESTADDRREQ]
39847 The socket is not connection-mode and no peer address is set.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

39848 [EINTR] A signal interrupted *send*( ) before any data was transmitted.

39849 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.

39850 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

39851 [ENOTSOCK] The *socket* argument does not refer to a socket.

39852 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
39853 more of the values set in *flags*.

39854 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
39855 no longer connected. In the latter case, and if the socket is of type
39856 SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.

39857 The *send*( ) function may fail if:

39858 [EACCES] The calling process does not have the appropriate privileges.

39859 [EIO] An I/O error occurred while reading from or writing to the file system.

39860 [ENETDOWN] The local network interface used to reach the destination is down.

39861 [ENETUNREACH]
39862 No route to the network is present.

39863 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

39864 **EXAMPLES**

39865 None.

39866 **APPLICATION USAGE**

39867 The *send*( ) function is equivalent to *sendto*( ) with a null pointer *dest_len* argument, and to *write*( )
39868 if no flags are used.

39869 **RATIONALE**

39870 None.

39871 **FUTURE DIRECTIONS**

39872 None.

39873 **SEE ALSO**

39874 *connect*( ), *getsockopt*( ), *poll*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *select*( ), *sendmsg*( ), *sendto*( ),
39875 *setsockopt*( ), *shutdown*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
39876 **<sys/socket.h>**

39877 **CHANGE HISTORY**

39878 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sendmsg — send a message on a socket using a message structure

SYNOPSIS

```
#include <sys/socket.h>

ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
```

DESCRIPTION

The *sendmsg*( ) function shall send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message shall be sent to the address specified by **msghdr**. If the socket is connection-mode, the destination address in **msghdr** shall be ignored.

The *sendmsg*( ) function takes the following arguments:

socket Specifies the socket file descriptor.

message Points to a **msghdr** structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The *msg_flags* member is ignored.

flags Specifies the type of message transmission. The application may specify 0 or the following flag:

MSG_EOR Terminates a record (if supported by the protocol).

MSG_OOB Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-band data are protocol-specific.

The *msg_iov* and *msg_iovlen* fields of *message* specify zero or more buffers containing the data to be sent. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by *msg_iov* is sent in turn.

Successful completion of a call to *sendmsg*( ) does not guarantee delivery of the message. A return value of –1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, the *sendmsg*( ) function shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, the *sendmsg*( ) function shall fail.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendmsg*( ) shall fail if the SO_BROADCAST option is not set for the socket.

The socket in use may require the process to have appropriate privileges to use the *sendmsg*( ) function.

RETURN VALUE

Upon successful completion, *sendmsg*( ) shall return the number of bytes sent. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sendmsg*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]
The socket's file descriptor is marked O_NONBLOCK and the requested

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

operation would block.

[EAFNOSUPPORT]
Addresses in the specified address family cannot be used with this socket.

[EBADF] The *socket* argument is not a valid file descriptor.

[ECONNRESET] A connection was forcibly closed by a peer.

[EINTR] A signal interrupted *sendmsg*( ) before any data was transmitted.

[EINVAL] The sum of the *iov_len* values overflows an **ssize_t**.

[EMSGSIZE] The message is too large to be sent all at once (as the socket requires), or the *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less than or equal to 0 or is greater than {IOV_MAX}.

[ENOTCONN] The socket is connection-mode but is not connected.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or more of the values set in *flags*.

[EPIPE] The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.

If the address family of the socket is AF_UNIX, then *sendmsg*( ) shall fail if:

[EIO] An I/O error occurred while reading from or writing to the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the pathname in the socket address.

[ENAMETOOLONG]
A component of a pathname exceeded {NAME_MAX} characters, or an entire pathname exceeded {PATH_MAX} characters.

[ENOENT] A component of the pathname does not name an existing file or the path name is an empty string.

[ENOTDIR] A component of the path prefix of the pathname in the socket address is not a directory.

The *sendmsg*( ) function may fail if:

[EACCES] Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

[EDESTADDRREQ]
The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

[EHOSTUNREACH]
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

[EIO] An I/O error occurred while reading from or writing to the file system.

[EISCONN] A destination address was specified and the socket is already connected.

[ENETDOWN] The local network interface used to reach the destination is down.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

39963 [ENETUNREACH]
39964 No route to the network is present.

39965 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

39966 [ENOMEM] Insufficient memory was available to fulfill the request.

39967 If the address family of the socket is AF_UNIX, then *sendmsg*( ) may fail if:

39968 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
39969 resolution of the pathname in the socket address.

39970 [ENAMETOOLONG]
39971 Pathname resolution of a symbolic link produced an intermediate result
39972 whose length exceeds {PATH_MAX}.

39973 **EXAMPLES**

39974 Done.

39975 **APPLICATION USAGE**

39976 The *select*( ) and *poll*( ) functions can be used to determine when it is possible to send more data.

39977 **RATIONALE**

39978 None.

39979 **FUTURE DIRECTIONS**

39980 None.

39981 **SEE ALSO**

39982 *getsockopt*( ), *poll*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *select*( ), *send*( ), *sendto*( ), *setsockopt*( ),
39983 *shutdown*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

39984 **CHANGE HISTORY**

39985 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39986 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
39987 [ELOOP] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sendto — send a message on a socket

SYNOPSIS

```
#include <sys/socket.h>

ssize_t sendto(int socket, const void *message, size_t length,
    int flags, const struct sockaddr *dest_addr,
    socklen_t dest_len);
```

DESCRIPTION

The *sendto*( ) function shall send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message shall be sent to the address specified by *dest_addr*. If the socket is connection-mode, *dest_addr* shall be ignored.

The *sendto*( ) function takes the following arguments:

socket Specifies the socket file descriptor.

message Points to a buffer containing the message to be sent.

length Specifies the size of the message in bytes.

flags Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:

- MSG_EOR Terminates a record (if supported by the protocol).
- MSG_OOB Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

dest_addr Points to a **sockaddr** structure containing the destination address. The length and format of the address depend on the address family of the socket.

dest_len Specifies the length of the **sockaddr** structure pointed to by the *dest_addr* argument.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendto*( ) shall fail if the SO_BROADCAST option is not set for the socket.

The *dest_addr* argument specifies the address of the target. The *length* argument specifies the length of the message.

Successful completion of a call to *sendto*( ) does not guarantee delivery of the message. A return value of –1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, *sendto*( ) shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, *sendto*( ) shall fail.

The socket in use may require the process to have appropriate privileges to use the *sendto*( ) function.

RETURN VALUE

Upon successful completion, *sendto*( ) shall return the number of bytes sent. Otherwise, –1 shall be returned and *errno* set to indicate the error.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ERRORS

The *sendto*( ) function shall fail if:

[EAFNOSUPPORT]
Addresses in the specified address family cannot be used with this socket.

[EAGAIN] or [EWOULDBLOCK]
The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.

[EBADF] The *socket* argument is not a valid file descriptor.

[ECONNRESET] A connection was forcibly closed by a peer.

[EINTR] A signal interrupted *sendto*( ) before any data was transmitted.

[EMSGSIZE] The message is too large to be sent all at once, as the socket requires.

[ENOTCONN] The socket is connection-mode but is not connected.

[ENOTSOCK] The *socket* argument does not refer to a socket.

[EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or more of the values set in *flags*.

[EPIPE] The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.

If the address family of the socket is AF_UNIX, then *sendto*( ) shall fail if:

[EIO] An I/O error occurred while reading from or writing to the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the pathname in the socket address.

[ENAMETOOLONG]
A component of a pathname exceeded {NAME_MAX} characters, or an entire pathname exceeded {PATH_MAX} characters.

[ENOENT] A component of the pathname does not name an existing file or the pathname is an empty string.

[ENOTDIR] A component of the path prefix of the pathname in the socket address is not a directory.

The *sendto*( ) function may fail if:

[EACCES] Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

[EDESTADDRREQ]
The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

[EHOSTUNREACH]
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

[EINVAL] The *dest_len* argument is not a valid length for the address family.

[EIO] An I/O error occurred while reading from or writing to the file system.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40068 [EISCONN] A destination address was specified and the socket is already connected. This
40069 error may or may not be returned for connection mode sockets.

40070 [ENETDOWN] The local network interface used to reach the destination is down.

40071 [ENETUNREACH]
40072 No route to the network is present.

40073 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

40074 [ENOMEM] Insufficient memory was available to fulfill the request.

40075 If the address family of the socket is AF_UNIX, then *sendto*( ) may fail if:

40076 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
40077 resolution of the pathname in the socket address.

40078 [ENAMETOOLONG]
40079 Pathname resolution of a symbolic link produced an intermediate result
40080 whose length exceeds {PATH_MAX}.

40081 **EXAMPLES**

40082 None.

40083 **APPLICATION USAGE**

40084 The *select*( ) and *poll*( ) functions can be used to determine when it is possible to send more data.

40085 **RATIONALE**

40086 None.

40087 **FUTURE DIRECTIONS**

40088 None.

40089 **SEE ALSO**

40090 *getsockopt*( ), *poll*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *select*( ), *send*( ), *sendmsg*( ), *setsockopt*( ),
40091 *shutdown*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

40092 **CHANGE HISTORY**

40093 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

40094 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
40095 [ELOOP] error condition is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

setbuf — assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

void setbuf(FILE *restrict stream, char *restrict buf);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

Except that it returns no value, the function call:

```
setbuf(stream, buf)
```

shall be equivalent to:

```
setvbuf(stream, buf, _IOFBF, BUFSIZ)
```

if *buf* is not a null pointer, or to:

```
setvbuf(stream, buf, _IONBF, BUFSIZ)
```

if *buf* is a null pointer.

RETURN VALUE

The *setbuf*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

A common source of error is allocating buffer space as an ‘‘automatic’’ variable in a code block, and then failing to close the stream in the same block.

With *setbuf*( ), allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ bytes are used for the buffer area.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fopen( ), *setvbuf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The prototype for *setbuf*( ) is updated for alignment with the ISO/IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40133 **NAME**

40134 setcontext — set current user context

40135 **SYNOPSIS**

40136 XSI
```
#include <ucontext.h>
```

40137
```
int setcontext(const ucontext_t *ucp);
```

40138

40139 **DESCRIPTION**

40140 Refer to *getcontext*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40141 **NAME**

40142 setegid — set the effective group ID

40143 **SYNOPSIS**

40144 `#include <unistd.h>`

40145 `int setegid(gid_t gid);`

40146 **DESCRIPTION**

40147 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate
40148 privileges, *setegid*( ) shall set the effective group ID of the calling process to *gid*; the real group
40149 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

40150 The *setegid*( ) function shall not affect the supplementary group list in any way.

40151 **RETURN VALUE**

40152 Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to
40153 indicate the error.

40154 **ERRORS**

40155 The *setegid*( ) function shall fail if:

40156 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
40157 implementation.

40158 [EPERM] The process does not have appropriate privileges and *gid* does not match the
40159 real group ID or the saved set-group-ID.

40160 **EXAMPLES**

40161 None.

40162 **APPLICATION USAGE**

40163 None.

40164 **RATIONALE**

40165 Refer to the RATIONALE section in *setuid*( ).

40166 **FUTURE DIRECTIONS**

40167 None.

40168 **SEE ALSO**

40169 *exec*, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( ), the
40170 Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

40171 **CHANGE HISTORY**

40172 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40173 **NAME**

40174 setenv — add or change environment variable

40175 **SYNOPSIS**

40176 CX
```
#include <stdlib.h>

int setenv(const char *envname, const char *envval, int overwrite);
```
40177

40178

40179 **DESCRIPTION**

40180 The *setenv*( ) function shall update or add a variable in the environment of the calling process.
40181 The *envname* argument points to a string containing the name of an environment variable to be
40182 added or altered. The environment variable shall be set to the value to which *envval* points. The
40183 function shall fail if *envname* points to a string which contains an '=' character. If the
40184 environment variable named by *envname* already exists and the value of *overwrite* is non-zero,
40185 the function shall return success and the environment shall be updated. If the environment
40186 variable named by *envname* already exists and the value of *overwrite* is zero, the function shall
40187 return success and the environment shall remain unchanged.

40188 If the application modifies *environ* or the pointers to which it points, the behavior of *setenv*( ) is
40189 undefined. The *setenv*( ) function shall update the list of pointers to which *environ* points.

40190 The strings described by *envname* and *envval* are copied by this function.

40191 The *setenv*( ) function need not be reentrant. A function that is not required to be reentrant is not
40192 required to be thread-safe.

40193 **RETURN VALUE**

40194 Upon successful completion, zero shall be returned. Otherwise, –1 shall be returned, *errno* set to
40195 indicate the error, and the environment shall be unchanged.

40196 **ERRORS**

40197 The *setenv*( ) function shall fail if:

40198 [EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a
40199 string containing an '=' character.

40200 [ENOMEM] Insufficient memory was available to add a variable or its value to the
40201 environment.

40202 **EXAMPLES**

40203 None.

40204 **APPLICATION USAGE**

40205 See *exec*, for restrictions on changing the environment in multi-threaded applications.

40206 **RATIONALE**

40207 Unanticipated results may occur if *setenv*( ) changes the external variable *environ*. In particular,
40208 if the optional *envp* argument to *main*( ) is present, it is not changed, and thus may point to an
40209 obsolete copy of the environment (as may any other copy of *environ*). However, other than the
40210 aforementioned restriction, the developers of IEEE Std 1003.1-2001 intended that the traditional
40211 method of walking through the environment by way of the *environ* pointer must be supported.

40212 It was decided that *setenv*( ) should be required by this revision because it addresses a piece of
40213 missing functionality, and does not impose a significant burden on the implementor.

40214 There was considerable debate as to whether the System V *putenv*( ) function or the BSD *setenv*( )
40215 function should be required as a mandatory function. The *setenv*( ) function was chosen because
40216 it permitted the implementation of the *unsetenv*( ) function to delete environmental variables,
40217 without specifying an additional interface. The *putenv*( ) function is available as an XSI

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

extension.

The standard developers considered requiring that *setenv*( ) indicate an error when a call to it would result in exceeding {ARG_MAX}. The requirement was rejected since the condition might be temporary, with the application eventually reducing the environment size. The ultimate success or failure depends on the size at the time of a call to *exec*, which returns an indication of this error condition.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getenv*( ), *unsetenv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in the APPLICATION USAGE and SEE ALSO sections.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40233 **NAME**

40234 seteuid — set effective user ID

40235 **SYNOPSIS**

40236 `#include <unistd.h>`

40237 `int seteuid(uid_t uid);`

40238 **DESCRIPTION**

40239 If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate
40240 privileges, *seteuid*( ) shall set the effective user ID of the calling process to *uid*; the real user ID
40241 and saved set-user-ID shall remain unchanged.

40242 The *seteuid*( ) function shall not affect the supplementary group list in any way.

40243 **RETURN VALUE**

40244 Upon successful completion, 0 shall be returned; otherwise, –1 shall be returned and *errno* set to
40245 indicate the error.

40246 **ERRORS**

40247 The *seteuid*( ) function shall fail if:

40248 [EINVAL] The value of the *uid* argument is invalid and is not supported by the
40249 implementation.

40250 [EPERM] The process does not have appropriate privileges and *uid* does not match the
40251 real group ID or the saved set-group-ID.

40252 **EXAMPLES**

40253 None.

40254 **APPLICATION USAGE**

40255 None.

40256 **RATIONALE**

40257 Refer to the RATIONALE section in *setuid*( ).

40258 **FUTURE DIRECTIONS**

40259 None.

40260 **SEE ALSO**

40261 *exec*, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( ), the
40262 Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

40263 **CHANGE HISTORY**

40264 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40265 **NAME**

40266 setgid — set-group-ID

40267 **SYNOPSIS**

40268
```
#include <unistd.h>
```

40269
```
int setgid(gid_t gid);
```

40270 **DESCRIPTION**

40271 If the process has appropriate privileges, *setgid*( ) shall set the real group ID, effective group ID,
40272 and the saved set-group-ID of the calling process to *gid*.

40273 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the
40274 saved set-group-ID, *setgid*( ) shall set the effective group ID to *gid*; the real group ID and saved
40275 set-group-ID shall remain unchanged.

40276 The *setgid*( ) function shall not affect the supplementary group list in any way.

40277 Any supplementary group IDs of the calling process shall remain unchanged.

40278 **RETURN VALUE**

40279 Upon successful completion, 0 is returned. Otherwise, –1 shall be returned and *errno* set to
40280 indicate the error.

40281 **ERRORS**

40282 The *setgid*( ) function shall fail if:

40283 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
40284 implementation.

40285 [EPERM] The process does not have appropriate privileges and *gid* does not match the
40286 real group ID or the saved set-group-ID.

40287 **EXAMPLES**

40288 None.

40289 **APPLICATION USAGE**

40290 None.

40291 **RATIONALE**

40292 Refer to the RATIONALE section in *setuid*( ).

40293 **FUTURE DIRECTIONS**

40294 None.

40295 **SEE ALSO**

40296 *exec*, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setregid*( ), *setreuid*( ), *setuid*( ), the
40297 Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

40298 **CHANGE HISTORY**

40299 First released in Issue 1. Derived from Issue 1 of the SVID.

40300 **Issue 6**

40301 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

40302 The following new requirements on POSIX implementations derive from alignment with the
40303 Single UNIX Specification:

40304 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
40305 required for conforming implementations of previous POSIX specifications, it was not
40306 required for UNIX applications.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- Functionality associated with _POSIX_SAVED_IDS is now mandated. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effects of *setgid*( ) in processes without appropriate privileges are changed.
- A requirement that the supplementary group list is not affected is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

setgrent — reset the group database to the first entry

SYNOPSIS

XSI

```
#include <grp.h>

void setgrent(void);
```

DESCRIPTION

Refer to *endgrent*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40320 **NAME**

40321 sethostent — network host database functions

40322 **SYNOPSIS**

40323 `#include <netdb.h>`

40324 `void sethostent(int` *`stayopen`*`);`

40325 **DESCRIPTION**

40326 Refer to *endhostent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

40327 **NAME**

40328 setitimer — set the value of an interval timer

40329 **SYNOPSIS**

40330 XSI
```
#include <sys/time.h>
```

40331
```
int setitimer(int which, const struct itimerval *restrict value,
```
40332
```
    struct itimerval *restrict ovalue);
```
40333

40334 **DESCRIPTION**

40335 Refer to *getitimer*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40336 **NAME**

40337 setjmp — set jump point for a non-local goto

40338 **SYNOPSIS**

40339 `#include <setjmp.h>`

40340 `int setjmp(jmp_buf env);`

40341 **DESCRIPTION**

40342 CX The functionality described on this reference page is aligned with the ISO C standard. Any
40343 conflict between the requirements described here and the ISO C standard is unintentional. This
40344 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40345 A call to *setjmp*( ) shall save the calling environment in its *env* argument for later use by
40346 *longjmp*( ).

40347 It is unspecified whether *setjmp*( ) is a macro or a function. If a macro definition is suppressed in
40348 order to access an actual function, or a program defines an external identifier with the name
40349 *setjmp*, the behavior is undefined.

40350 An application shall ensure that an invocation of *setjmp*( ) appears in one of the following
40351 contexts only:

40352 - The entire controlling expression of a selection or iteration statement
40353 - One operand of a relational or equality operator with the other operand an integral constant
40354 expression, with the resulting expression being the entire controlling expression of a
40355 selection or iteration statement
40356 - The operand of a unary '!' operator with the resulting expression being the entire
40357 controlling expression of a selection or iteration
40358 - The entire expression of an expression statement (possibly cast to **void**)

40359 If the invocation appears in any other context, the behavior is undefined.

40360 **RETURN VALUE**

40361 If the return is from a direct invocation, *setjmp*( ) shall return 0. If the return is from a call to
40362 *longjmp*( ), *setjmp*( ) shall return a non-zero value.

40363 **ERRORS**

40364 No errors are defined.

40365 **EXAMPLES**

40366 None.

40367 **APPLICATION USAGE**

40368 In general, *sigsetjmp*( ) is more useful in dealing with errors and interrupts encountered in a low-
40369 level subroutine of a program.

40370 **RATIONALE**

40371 None.

40372 **FUTURE DIRECTIONS**

40373 None.

40374 **SEE ALSO**

40375 *longjmp*( ), *sigsetjmp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<setjmp.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40376 **CHANGE HISTORY**

40377 First released in Issue 1. Derived from Issue 1 of the SVID.

40378 **Issue 6**

40379 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

setkey — set encoding key (**CRYPT**)

SYNOPSIS

XSI
```
#include <stdlib.h>

void setkey(const char *key);
```

DESCRIPTION

The *setkey*( ) function provides access to an implementation-defined encoding algorithm. The argument of *setkey*( ) is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used with the algorithm to encode a string *block* passed to *encrypt*( ).

The *setkey*( ) function shall not change the setting of *errno* if successful. An application wishing to check for error situations should set *errno* to 0 before calling *setkey*( ). If *errno* is non-zero on return, an error has occurred.

The *setkey*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

No values are returned.

ERRORS

The *setkey*( ) function shall fail if:

[ENOSYS] The functionality is not supported on this implementation.

EXAMPLES

None.

APPLICATION USAGE

Decoding need not be implemented in all environments. This is related to government restrictions in some countries on encryption and decryption routines. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of *encrypt*( ) does encoding but not decoding.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

crypt( ), *encrypt*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40419 **NAME**

40420 setlocale — set program locale

40421 **SYNOPSIS**

40422 `#include <locale.h>`

40423 `char *setlocale(int category, const char *locale);`

40424 **DESCRIPTION**

40425 CX The functionality described on this reference page is aligned with the ISO C standard. Any
40426 conflict between the requirements described here and the ISO C standard is unintentional. This
40427 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40428 The *setlocale*( ) function selects the appropriate piece of the program's locale, as specified by the
40429 *category* and *locale* arguments, and may be used to change or query the program's entire locale or
40430 portions thereof. The value *LC_ALL* for *category* names the program's entire locale; other values
40431 for *category* name only a part of the program's locale:

40432 *LC_COLLATE* Affects the behavior of regular expressions and the collation functions.

40433 *LC_CTYPE* Affects the behavior of regular expressions, character classification, character
40434 conversion functions, and wide-character functions.

40435 CX *LC_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or
40436 negative responses.

40437 XSI It also affects what strings are given by commands and utilities as affirmative
40438 or negative responses, and the content of messages.

40439 *LC_MONETARY* Affects the behavior of functions that handle monetary values.

40440 *LC_NUMERIC* Affects the behavior of functions that handle numeric values.

40441 *LC_TIME* Affects the behavior of the time conversion functions.

40442 The *locale* argument is a pointer to a character string containing the required setting of *category*.
40443 The contents of this string are implementation-defined. In addition, the following preset values
40444 of *locale* are defined for all settings of *category*:

40445 CX "POSIX" Specifies the minimal environment for C-language translation called the
40446 POSIX locale. If *setlocale*( ) is not invoked, the POSIX locale is the default at
40447 entry to *main*( ).

40448 "C" Equivalent to `"POSIX"`.

40449 CX "" Specifies an implementation-defined native environment. This corresponds to
40450 the value of the associated environment variables, *LC_** and *LANG*; see the
40451 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale and the
40452 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment
40453 Variables.

40454 A null pointer Used to direct *setlocale*( ) to query the current internationalized environment
40455 and return the name of the locale.

40456 THR The locale state is common to all threads within a process.

40457 **RETURN VALUE**

40458 Upon successful completion, *setlocale*( ) shall return the string associated with the specified
40459 category for the new locale. Otherwise, *setlocale*( ) shall return a null pointer and the program's
40460 locale is not changed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A null pointer for *locale* causes *setlocale*( ) to return a pointer to the string associated with the *category* for the program's current locale. The program's locale shall not be changed.

The string returned by *setlocale*( ) is such that a subsequent call with that string and its associated *category* shall restore that part of the program's locale. The application shall not modify the string returned which may be overwritten by a subsequent call to *setlocale*( ).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The following code illustrates how a program can initialize the international environment for one language, while selectively modifying the program's locale such that regular expressions and string operations can be applied to text recorded in a different language:

```
setlocale(LC_ALL, "De");
setlocale(LC_COLLATE, "Fr@dict");
```

Internationalized programs must call *setlocale*( ) to initiate a specific language operation. This can be done by calling *setlocale*( ) as follows:

```
setlocale(LC_ALL, "");
```

Changing the setting of *LC_MESSAGES* has no effect on catalogs that have already been opened by calls to *catopen*( ).

RATIONALE

The ISO C standard defines a collection of functions to support internationalization. One of the most significant aspects of these functions is a facility to set and query the *international environment*. The international environment is a repository of information that affects the behavior of certain functionality, namely:

1. Character handling
2. Collating
3. Date / time formatting
4. Numeric editing
5. Monetary formatting
6. Messaging

The *setlocale*( ) function provides the application developer with the ability to set all or portions, called *categories*, of the international environment. These categories correspond to the areas of functionality mentioned above. The syntax for *setlocale*( ) is as follows:

```
char *setlocale(int category, const char *locale);
```

where *category* is the name of one of following categories, namely:

LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In addition, a special value called *LC_ALL* directs *setlocale*( ) to set all categories.

There are two primary uses of *setlocale*( ):

1. Querying the international environment to find out what it is set to
2. Setting the international environment, or *locale*, to a specific value

The behavior of *setlocale*( ) in these two areas is described below. Since it is difficult to describe the behavior in words, examples are used to illustrate the behavior of specific uses.

To query the international environment, *setlocale*( ) is invoked with a specific category and the NULL pointer as the locale. The NULL pointer is a special directive to *setlocale*( ) that tells it to query rather than set the international environment. The following syntax is used to query the name of the international environment:

```
setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
    LC_NUMERIC, LC_TIME},(char *) NULL);
```

The *setlocale*( ) function shall return the string corresponding to the current international environment. This value may be used by a subsequent call to *setlocale*( ) to reset the international environment to this value. However, it should be noted that the return value from *setlocale*( ) may be a pointer to a static area within the function and is not guaranteed to remain unchanged (that is, it may be modified by a subsequent call to *setlocale*( )). Therefore, if the purpose of calling *setlocale*( ) is to save the value of the current international environment so it can be changed and reset later, the return value should be copied to an array of **char** in the calling program.

There are three ways to set the international environment with *setlocale*( ):

setlocale(*category*, *string*)
: This usage sets a specific *category* in the international environment to a specific value corresponding to the value of the *string*. A specific example is provided below:

```
setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

: In this example, all categories of the international environment are set to the locale corresponding to the string `"fr_FR.ISO-8859-1"`, or to the French language as spoken in France using the ISO / IEC 8859-1: 1998 standard codeset.

: If the string does not correspond to a valid locale, *setlocale*( ) shall return a NULL pointer and the international environment is not changed. Otherwise, *setlocale*( ) shall return the name of the locale just set.

setlocale(*category*, "C")
: The ISO C standard states that one locale must exist on all conforming implementations. The name of the locale is C and corresponds to a minimal international environment needed to support the C programming language.

setlocale(*category*, "")
: This sets a specific category to an implementation-defined default. This corresponds to the value of the environment variables.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *isalnum*( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *iswalnum*( ), *iswalpha*( ), *iswblank*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *isxdigit*( ),

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40547 *localeconv*( ), *mblen*( ), *mbstowcs*( ), *mbtowc*( ), *nl_langinfo*( ), *printf*( ), *scanf*( ), *setlocale*( ), *strcoll*( ),
40548 *strerror*( ), *strfmon*( ), *strtod*( ), *strxfrm*( ), *tolower*( ), *toupper*( ), *towlower*( ), *towupper*( ), *wcscoll*( ),
40549 *wcstod*( ), *wcstombs*( ), *wcsxfrm*( ), *wctomb*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
40550 **<langinfo.h>**, **<locale.h>**

40551 **CHANGE HISTORY**

40552 First released in Issue 3.

40553 **Issue 5**

40554 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

40555 **Issue 6**

40556 Extensions beyond the ISO C standard are marked.

40557 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40558 **NAME**

40559 setlogmask — set the log priority mask

40560 **SYNOPSIS**

40561 XSI
```
#include <syslog.h>
```

40562
```
int setlogmask(int maskpri);
```

40563

40564 **DESCRIPTION**

40565 Refer to *closelog*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40566 **NAME**

40567 setnetent — network database function

40568 **SYNOPSIS**

40569
```
#include <netdb.h>
```

40570
```
void setnetent(int stayopen);
```

40571 **DESCRIPTION**

40572 Refer to *endnetent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

setpgid — set process group ID for job control

SYNOPSIS

```
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
```

DESCRIPTION

The *setpgid*( ) function shall either join an existing process group or create a new process group within the session of the calling process. The process group ID of a session leader shall not change. Upon successful completion, the process group ID of the process with a process ID that matches *pid* shall be set to *pgid*. As a special case, if *pid* is 0, the process ID of the calling process shall be used. Also, if *pgid* is 0, the process ID of the indicated process shall be used.

RETURN VALUE

Upon successful completion, *setpgid*( ) shall return 0; otherwise, –1 shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *setpgid*( ) function shall fail if:

[EACCES] The value of the *pid* argument matches the process ID of a child process of the calling process and the child process has successfully executed one of the *exec* functions.

[EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by the implementation.

[EPERM] The process indicated by the *pid* argument is a session leader.

[EPERM] The value of the *pid* argument matches the process ID of a child process of the calling process and the child process is not in the same session as the calling process.

[EPERM] The value of the *pgid* argument is valid but does not match the process ID of the process indicated by the *pid* argument and there is no process with a process group ID that matches the value of the *pgid* argument in the same session as the calling process.

[ESRCH] The value of the *pid* argument does not match the process ID of the calling process or of a child process of the calling process.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *setpgid*( ) function shall group processes together for the purpose of signaling, placement in foreground or background, and other job control actions.

The *setpgid*( ) function is similar to the *setpgrp*( ) function of 4.2 BSD, except that 4.2 BSD allowed the specified new process group to assume any value. This presents certain security problems and is more flexible than necessary to support job control.

To provide tighter security, *setpgid*( ) only allows the calling process to join a process group already in use inside its session or create a new process group whose process group ID was equal to its process ID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When a job control shell spawns a new job, the processes in the job must be placed into a new process group via *setpgid*( ). There are two timing constraints involved in this action:

1. The new process must be placed in the new process group before the appropriate program is launched via one of the *exec* functions.
2. The new process must be placed in the new process group before the shell can correctly send signals to the new process group.

To address these constraints, the following actions are performed. The new processes call *setpgid*( ) to alter their own process groups after *fork*( ) but before *exec*. This satisfies the first constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of *vfork*( ); that is, the shell is suspended until the child has completed the *exec*, thus ensuring that the child has completed the *setpgid*( ). A new version of *fork*( ) with this same synchronization property was considered, but it was decided instead to merely allow the parent shell process to adjust the process group of its child processes via *setpgid*( ). Both timing constraints are now satisfied by having both the parent shell and the child attempt to adjust the process group of the child process; it does not matter which succeeds first.

Since it would be confusing to an application to have its process group change after it began executing (that is, after *exec*), and because the child process would already have adjusted its process group before this, the [EACCES] error was added to disallow this.

One non-obvious use of *setpgid*( ) is to allow a job control shell to return itself to its original process group (the one in effect when the job control shell was executed). A job control shell does this before returning control back to its parent when it is terminating or suspending itself as a way of restoring its job control ''state'' back to what its parent would expect. (Note that the original process group of the job control shell typically matches the process group of its parent, but this is not necessarily always the case.)

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getpgrp*( ), *setsid*( ), *tcsetpgrp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The *setpgid*( ) function is mandatory since _POSIX_JOB_CONTROL is required to be defined in this issue. This is a FIPS requirement.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in the DESCRIPTION from ''the process group ID of the indicated process shall be used'' to ''the process ID of the indicated process shall be used''. This change reverts the wording to as in the ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40661 **NAME**

40662 setpgrp — set the process group ID

40663 **SYNOPSIS**

40664 XSI
```
#include <unistd.h>
```

40665
```
pid_t setpgrp(void);
```

40666

40667 **DESCRIPTION**

40668 If the calling process is not already a session leader, *setpgrp*( ) sets the process group ID of the
40669 calling process to the process ID of the calling process. If *setpgrp*( ) creates a new session, then
40670 the new session has no controlling terminal.

40671 The *setpgrp*( ) function has no effect when the calling process is a session leader.

40672 **RETURN VALUE**

40673 Upon completion, *setpgrp*( ) shall return the process group ID.

40674 **ERRORS**

40675 No errors are defined.

40676 **EXAMPLES**

40677 None.

40678 **APPLICATION USAGE**

40679 None.

40680 **RATIONALE**

40681 None.

40682 **FUTURE DIRECTIONS**

40683 None.

40684 **SEE ALSO**

40685 *exec*, *fork*( ), *getpid*( ), *getsid*( ), *kill*( ), *setpgid*( ), *setsid*( ), the Base Definitions volume of
40686 IEEE Std 1003.1-2001, **<unistd.h>**

40687 **CHANGE HISTORY**

40688 First released in Issue 4, Version 2.

40689 **Issue 5**

40690 Moved from X/OPEN UNIX extension to BASE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40691 **NAME**

40692 setpriority — set the nice value

40693 **SYNOPSIS**

40694 XSI
```
#include <sys/resource.h>
```

40695
```
int setpriority(int which, id_t who, int nice);
```

40696

40697 **DESCRIPTION**

40698 Refer to *getpriority*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

40699 **NAME**

40700 setprotoent — network protocol database functions

40701 **SYNOPSIS**

40702 `#include <netdb.h>`

40703 `void setprotoent(int` *`stayopen`*`);`

40704 **DESCRIPTION**

40705 Refer to *endprotoent*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40706 **NAME**

40707 setpwent — user database function

40708 **SYNOPSIS**

40709 XSI
```
#include <pwd.h>
```

40710
```
void setpwent(void);
```

40711

40712 **DESCRIPTION**

40713 Refer to *endpwent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

setregid — set real and effective group IDs

SYNOPSIS

XSI
```
#include <unistd.h>

int setregid(gid_t rgid, gid_t egid);
```

DESCRIPTION

The *setregid*( ) function shall set the real and effective group IDs of the calling process.

If *rgid* is –1, the real group ID shall not be changed; if *egid* is –1, the effective group ID shall not be changed.

The real and effective group IDs may be set to different values in the same call.

Only a process with appropriate privileges can set the real group ID and the effective group ID to any valid value.

A non-privileged process can set either the real group ID to the saved set-group-ID from one of the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group ID.

Any supplementary group IDs of the calling process remain unchanged.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error, and neither of the group IDs are changed.

ERRORS

The *setregid*( ) function shall fail if:

[EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.

[EPERM] The process does not have appropriate privileges and a change other than changing the real group ID to the saved set-group-ID, or changing the effective group ID to the real group ID or the saved set-group-ID, was requested.

EXAMPLES

None.

APPLICATION USAGE

If a set-group-ID process sets its effective group ID to its real group ID, it can still set its effective group ID back to the saved set-group-ID.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setreuid*( ), *setuid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40755 **Issue 5**

40756 Moved from X/OPEN UNIX extension to BASE.

40757 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the
40758 *exec* family of functions, not just *execve*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

setreuid — set real and effective user IDs

SYNOPSIS

XSI
```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
```

DESCRIPTION

The *setreuid*( ) function shall set the real and effective user IDs of the current process to the values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is –1, the corresponding effective or real user ID of the current process shall be left unchanged.

A process with appropriate privileges can set either ID to any value. An unprivileged process can only set the effective user ID if the *euid* argument is equal to either the real, effective, or saved user ID of the process.

It is unspecified whether a process without appropriate privileges is permitted to change the real user ID to match the current real, effective, or saved set-user-ID of the process.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setreuid*( ) function shall fail if:

[EINVAL] The value of the *ruid* or *euid* argument is invalid or out-of-range.

[EPERM] The current process does not have appropriate privileges, and either an attempt was made to change the effective user ID to a value other than the real user ID or the saved set-user-ID or an attempt was made to change the real user ID to a value not permitted by the implementation.

EXAMPLES

Setting the Effective User ID to the Real User ID

The following example sets the effective user ID of the calling process to the real user ID, so that files created later will be owned by the current user.

```
#include <unistd.h>
#include <sys/types.h>
...
setreuid(getuid(), getuid());
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40799 **SEE ALSO**

40800 *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setuid*( ), the Base
40801 Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

40802 **CHANGE HISTORY**

40803 First released in Issue 4, Version 2.

40804 **Issue 5**

40805 Moved from X/OPEN UNIX extension to BASE.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

40806 **NAME**

40807 setrlimit — control maximum resource consumption

40808 **SYNOPSIS**

40809 XSI
```
#include <sys/resource.h>
```

40810
```
int setrlimit(int resource, const struct rlimit *rlp);
```

40811

40812 **DESCRIPTION**

40813 Refer to *getrlimit*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40814 **NAME**

40815 setservent — network services database functions

40816 **SYNOPSIS**

40817 `#include <netdb.h>`

40818 `void setservent(int` *`stayopen`*`);`

40819 **DESCRIPTION**

40820 Refer to *endservent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

setsid — create session and set process group ID

SYNOPSIS

```
#include <unistd.h>

pid_t setsid(void);
```

DESCRIPTION

The *setsid*( ) function shall create a new session, if the calling process is not a process group leader. Upon return the calling process shall be the session leader of this new session, shall be the process group leader of a new process group, and shall have no controlling terminal. The process group ID of the calling process shall be set equal to the process ID of the calling process. The calling process shall be the only process in the new process group and the only process in the new session.

RETURN VALUE

Upon successful completion, *setsid*( ) shall return the value of the new process group ID of the calling process. Otherwise, it shall return (**pid_t**)–1 and set *errno* to indicate the error.

ERRORS

The *setsid*( ) function shall fail if:

[EPERM] The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *setsid*( ) function is similar to the *setpgrp*( ) function of System V. System V, without job control, groups processes into process groups and creates new process groups via *setpgrp*( ); only one process group may be part of a login session.

Job control allows multiple process groups within a login session. In order to limit job control actions so that they can only affect processes in the same login session, this volume of IEEE Std 1003.1-2001 adds the concept of a session that is created via *setsid*( ). The *setsid*( ) function also creates the initial process group contained in the session. Additional process groups can be created via the *setpgid*( ) function. A System V process group would correspond to a POSIX System Interfaces session containing a single POSIX process group. Note that this function requires that the calling process not be a process group leader. The usual way to ensure this is true is to create a new process with *fork*( ) and have it call *setsid*( ). The *fork*( ) function guarantees that the process ID of the new process does not match any existing process group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

getsid( ), *setpgid*( ), *setpgrp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40863 **CHANGE HISTORY**

40864 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

40865 **Issue 6**

40866 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

40867 The following new requirements on POSIX implementations derive from alignment with the
40868 Single UNIX Specification:

40869 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
40870 required for conforming implementations of previous POSIX specifications, it was not
40871 required for UNIX applications.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

setsockopt — set the socket options

SYNOPSIS

```
#include <sys/socket.h>

int setsockopt(int socket, int level, int option_name,
    const void *option_value, socklen_t option_len);
```

DESCRIPTION

The *setsockopt*( ) function shall set the option specified by the *option_name* argument, at the protocol level specified by the *level* argument, to the value pointed to by the *option_value* argument for the socket associated with the file descriptor specified by the *socket* argument.

The *level* argument specifies the protocol level at which the option resides. To set options at the socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply the appropriate *level* identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO_TCP as defined in the **<netinet/in.h>** header.

The *option_name* argument specifies a single option to set. The *option_name* argument and any specified options are passed uninterpreted to the appropriate protocol module for interpretations. The **<sys/socket.h>** header defines the socket-level options. The options are as follows:

SO_DEBUG — Turns on recording of debugging information. This option enables or disables debugging in the underlying protocol modules. This option takes an **int** value. This is a Boolean option.

SO_BROADCAST — Permits sending of broadcast messages, if this is supported by the protocol. This option takes an **int** value. This is a Boolean option.

SO_REUSEADDR — Specifies that the rules used in validating addresses supplied to *bind*( ) should allow reuse of local addresses, if this is supported by the protocol. This option takes an **int** value. This is a Boolean option.

SO_KEEPALIVE — Keeps connections active by enabling the periodic transmission of messages, if this is supported by the protocol. This option takes an **int** value.

If the connected socket fails to respond to these messages, the connection is broken and threads writing to that socket are notified with a SIGPIPE signal. This is a Boolean option.

SO_LINGER — Lingers on a *close*( ) if data is present. This option controls the action taken when unsent messages queue on a socket and *close*( ) is performed. If SO_LINGER is set, the system shall block the process during *close*( ) until it can transmit the data or until the time expires. If SO_LINGER is not specified, and *close*( ) is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option takes a **linger** structure, as defined in the **<sys/socket.h>** header, to specify the state of the option and linger interval.

SO_OOBINLINE — Leaves received out-of-band data (data marked urgent) inline. This option takes an **int** value. This is a Boolean option.

SO_SNDBUF — Sets send buffer size. This option takes an **int** value.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SO_RCVBUF — Sets receive buffer size. This option takes an **int** value.

SO_DONTROUTE — Requests that outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an **int** value. This is a Boolean option.

SO_RCVLOWAT — Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option takes an **int** value. Note that not all implementations allow this option to be set.

SO_RCVTIMEO — Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or *errno* set to [EAGAIN] or [EWOULDBLOCK] if no data is received. The default for this option is zero, which indicates that a receive operation shall not time out. This option takes a **timeval** structure. Note that not all implementations allow this option to be set.

SO_SNDLOWAT — Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an **int** value. Note that not all implementations allow this option to be set.

SO_SNDTIMEO — Sets the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with *errno* set to [EAGAIN] or [EWOULDBLOCK] if no data is sent. The default for this option is zero, which indicates that a send operation shall not time out. This option stores a **timeval** structure. Note that not all implementations allow this option to be set.

For Boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.

Options at other protocol levels vary in format and name.

RETURN VALUE

Upon successful completion, *setsockopt*( ) shall return 0. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setsockopt*( ) function shall fail if:

[EBADF] — The *socket* argument is not a valid file descriptor.

[EDOM] — The send and receive timeout values are too big to fit into the timeout fields in the socket structure.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EINVAL] The specified option is invalid at the specified socket level or the socket has been shut down.

[EISCONN] The socket is already connected, and a specified option cannot be set while the socket is connected.

[ENOPROTOOPT] The option is not supported by the protocol.

[ENOTSOCK] The *socket* argument does not refer to a socket.

The *setsockopt*( ) function may fail if:

[ENOMEM] There was insufficient memory available for the operation to complete.

[ENOBUFS] Insufficient resources are available in the system to complete the call.

EXAMPLES

None.

APPLICATION USAGE

The *setsockopt*( ) function provides an application program with the means to control socket behavior. An application program can use *setsockopt*( ) to allocate buffer space, control timeouts, or permit socket data broadcasts. The **<sys/socket.h>** header defines the socket-level options available to *setsockopt*( ).

Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10 (on page 58), *bind*( ), *endprotoent*( ), *getsockopt*( ), *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<netinet/in.h>**, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40991 **NAME**

40992 setstate — switch pseudo-random number generator state arrays

40993 **SYNOPSIS**

40994 XSI
```
#include <stdlib.h>

char *setstate(const char *state);
```
40995

40996

40997 **DESCRIPTION**

40998 Refer to *initstate*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

40999 **NAME**

41000 setuid — set user ID

41001 **SYNOPSIS**

41002 `#include <unistd.h>`

41003 `int setuid(uid_t uid);`

41004 **DESCRIPTION**

41005 If the process has appropriate privileges, *setuid*( ) shall set the real user ID, effective user ID, and
41006 the saved set-user-ID of the calling process to *uid*.

41007 If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the
41008 saved set-user-ID, *setuid*( ) shall set the effective user ID to *uid*; the real user ID and saved set-
41009 user-ID shall remain unchanged.

41010 The *setuid*( ) function shall not affect the supplementary group list in any way.

41011 **RETURN VALUE**

41012 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
41013 indicate the error.

41014 **ERRORS**

41015 The *setuid*( ) function shall fail, return –1, and set *errno* to the corresponding value if one or more
41016 of the following are true:

41017 [EINVAL] The value of the *uid* argument is invalid and not supported by the
41018 implementation.

41019 [EPERM] The process does not have appropriate privileges and *uid* does not match the
41020 real user ID or the saved set-user-ID.

41021 **EXAMPLES**

41022 None.

41023 **APPLICATION USAGE**

41024 None.

41025 **RATIONALE**

41026 The various behaviors of the *setuid*( ) and *setgid*( ) functions when called by non-privileged
41027 processes reflect the behavior of different historical implementations. For portability, it is
41028 recommended that new non-privileged applications use the *seteuid*( ) and *setegid*( ) functions
41029 instead.

41030 The saved set-user-ID capability allows a program to regain the effective user ID established at
41031 the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the
41032 effective group ID established at the last *exec* call. These capabilities are derived from System V.
41033 Without them, a program might have to run as superuser in order to perform the same
41034 functions, because superuser can write on the user's files. This is a problem because such a
41035 program can write on any user's files, and so must be carefully written to emulate the
41036 permissions of the calling process properly. In System V, these capabilities have traditionally
41037 been implemented only via the *setuid*( ) and *setgid*( ) functions for non-privileged processes. The
41038 fact that the behavior of those functions was different for privileged processes made them
41039 difficult to use. The POSIX.1-1990 standard defined the *setuid*( ) function to behave differently
41040 for privileged and unprivileged users. When the caller had the appropriate privilege, the
41041 function set the calling process' real user ID, effective user ID, and saved set-user ID on
41042 implementations that supported it. When the caller did not have the appropriate privilege, the
41043 function set only the effective user ID, subject to permission checks. The former use is generally
41044 needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

scope of IEEE Std 1003.1-2001. These utilities wish to change the user ID irrevocably to a new value, generally that of an unprivileged user. The latter use is needed for conforming applications that are installed with the set-user-ID bit and need to perform operations using the real user ID.

IEEE Std 1003.1-2001 augments the latter functionality with a mandatory feature named _POSIX_SAVED_IDS. This feature permits a set-user-ID application to switch its effective user ID back and forth between the values of its *exec*-time real user ID and effective user ID. Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this feature to work properly when it happened to be executed with the (implementation-defined) appropriate privilege. Furthermore, the application did not even have a means to tell whether it had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by IEEE Std 1003.1-2001. However, there are implementors who have been reluctant to support it given the limitation described above.

The 4.3BSD system handles the problem by supporting separate functions: *setuid*( ) (which always sets both the real and effective user IDs, like *setuid*( ) in IEEE Std 1003.1-2001 for privileged users), and *seteuid*( ) (which always sets just the effective user ID, like *setuid*( ) in IEEE Std 1003.1-2001 for non-privileged users). This separation of functionality into distinct functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of switching the effective user ID back and forth via *setreuid*( ), which permits reversing the real and effective user IDs. This model seems less desirable than the saved set-user-ID because the real user ID changes as a side effect. The current 4.4BSD includes saved effective IDs and uses them for *seteuid*( ) and *setegid*( ) as described above. The *setreuid*( ) and *setregid*( ) functions will be deprecated or removed.

The solution here is:

- Require that all implementations support the functionality of the saved set-user-ID, which is set by the *exec* functions and by privileged calls to *setuid*( ).
- Add the *seteuid*( ) and *setegid*( ) functions as portable alternatives to *setuid*( ) and *setgid*( ) for non-privileged and privileged processes.

Historical systems have provided two mechanisms for a set-user-ID process to change its effective user ID to be the same as its real user ID in such a way that it could return to the original effective user ID: the use of the *setuid*( ) function in the presence of a saved set-user-ID, or the use of the BSD *setreuid*( ) function, which was able to swap the real and effective user IDs. The changes included in IEEE Std 1003.1-2001 provide a new mechanism using *seteuid*( ) in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid*( ) mechanism will have a saved set-user-ID for each process, and most of the behavior controlled by _POSIX_SAVED_IDS has been changed to agree with the case where the option was defined. The *kill*( ) function is an exception. Implementors of the new *seteuid*( ) mechanism will generally be required to maintain compatibility with the older mechanisms previously supported by their systems. However, compatibility with this use of *setreuid*( ) and with the _POSIX_SAVED_IDS behavior of *kill*( ) is unfortunately complicated. If an implementation with a saved set-user-ID allows a process to use *setreuid*( ) to swap its real and effective user IDs, but were to leave the saved set-user-ID unmodified, the process would then have an effective user ID equal to the original real user ID, and both real and saved set-user-ID would be equal to the original effective user ID. In that state, the real user would be unable to kill the process, even though the effective user ID of the process matches that of the real user, if the *kill*( ) behavior of _POSIX_SAVED_IDS was used. This is obviously not acceptable. The alternative choice, which is used in at least one implementation, is to change the saved set-user-ID to the effective user ID during most calls to *setreuid*( ). The standard developers considered that alternative to be less correct than the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

retention of the old behavior of *kill*( ) in such systems. Current conforming applications shall accommodate either behavior from *kill*( ), and there appears to be no strong reason for *kill*( ) to check the saved set-user-ID rather than the effective user ID.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The functionality associated with _POSIX_SAVED_IDS is now mandatory. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effects of *setuid*( ) in processes without appropriate privileges are changed.
- A requirement that the supplementary group list is not affected is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

41116 **NAME**

41117 setutxent — reset the user accounting database to the first entry

41118 **SYNOPSIS**

41119 XSI
```
#include <utmpx.h>
```

41120
```
void setutxent(void);
```

41121

41122 **DESCRIPTION**

41123 Refer to *endutxent*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

setvbuf — assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

int setvbuf(FILE *restrict stream, char *restrict buf, int type,
    size_t size);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *setvbuf*( ) function may be used after the stream pointed to by *stream* is associated with an open file but before any other operation (other than an unsuccessful call to *setvbuf*( )) is performed on the stream. The argument *type* determines how *stream* shall be buffered, as follows:

- {_IOFBF} shall cause input/output to be fully buffered.
- {_IOLBF} shall cause input/output to be line buffered.
- {_IONBF} shall cause input/output to be unbuffered.

If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by *setvbuf*( ) and the argument *size* specifies the size of the array; otherwise, *size* may determine the size of a buffer allocated by the *setvbuf*( ) function. The contents of the array at any time are unspecified.

For information about streams, see Section 2.5 (on page 34).

RETURN VALUE

Upon successful completion, *setvbuf*( ) shall return 0. Otherwise, it shall return a non-zero value CX if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to indicate the error.

ERRORS

The *setvbuf*( ) function may fail if:

CX [EBADF] The file descriptor underlying *stream* is not valid.

EXAMPLES

None.

APPLICATION USAGE

A common source of error is allocating buffer space as an ‘‘automatic’’ variable in a code block, and then failing to close the stream in the same block.

With *setvbuf*( ), allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are used for the buffer area.

Applications should note that many implementations only provide line buffering on input from terminal devices.

RATIONALE

None.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

41164 **FUTURE DIRECTIONS**

41165 None.

41166 **SEE ALSO**

41167 Section 2.5 (on page 34), *fopen*( ), *setbuf*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
41168 **<stdio.h>**

41169 **CHANGE HISTORY**

41170 First released in Issue 1. Derived from Issue 1 of the SVID.

41171 **Issue 6**

41172 Extensions beyond the ISO C standard are marked.

41173 The *setvbuf*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

shm_open — open a shared memory object (**REALTIME**)

SYNOPSIS

SHM

```
#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);
```

DESCRIPTION

The *shm_open*( ) function shall establish a connection between a shared memory object and a file descriptor. It shall create an open file description that refers to the shared memory object and a file descriptor that refers to that open file description. The file descriptor is used by other functions to refer to that shared memory object. The *name* argument points to a string naming a shared memory object. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname. If *name* begins with the slash character, then processes calling *shm_open*( ) with the same value of *name* refer to the same shared memory object, as long as that name has not been removed. If *name* does not begin with the slash character, the effect is implementation-defined. The interpretation of slash characters other than the leading slash character in *name* is implementation-defined.

If successful, *shm_open*( ) shall return a file descriptor for the shared memory object that is the lowest numbered file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor does not share it with any other processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file descriptor is set.

The file status flags and file access modes of the open file description are according to the value of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the **<fcntl.h>** header. Applications specify exactly one of the first two values (access modes) below in the value of *oflag*:

O_RDONLY Open for read access only.

O_RDWR Open for read or write access.

Any combination of the remaining flags may be specified in the value of *oflag*:

O_CREAT If the shared memory object exists, this flag has no effect, except as noted under O_EXCL below. Otherwise, the shared memory object is created; the user ID of the shared memory object shall be set to the effective user ID of the process; the group ID of the shared memory object is set to a system default group ID or to the effective group ID of the process. The permission bits of the shared memory object shall be set to the value of the *mode* argument except those set in the file mode creation mask of the process. When bits in *mode* other than the file permission bits are set, the effect is unspecified. The *mode* argument does not affect whether the shared memory object is opened for reading, for writing, or for both. The shared memory object has a size of zero.

O_EXCL If O_EXCL and O_CREAT are set, *shm_open*( ) fails if the shared memory object exists. The check for the existence of the shared memory object and the creation of the object if it does not exist is atomic with respect to other processes executing *shm_open*( ) naming the same shared memory object with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

O_TRUNC If the shared memory object exists, and it is successfully opened O_RDWR, the object shall be truncated to zero length and the mode and owner shall be unchanged by this function call. The result of using O_TRUNC with O_RDONLY is undefined.

When a shared memory object is created, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other references are gone. It is unspecified whether the name and shared memory object state remain valid after a system reboot.

RETURN VALUE

Upon successful completion, the *shm_open*( ) function shall return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *shm_open*( ) function shall fail if:

[EACCES] The shared memory object exists and the permissions specified by *oflag* are denied, or the shared memory object does not exist and permission to create the shared memory object is denied, or O_TRUNC is specified and write permission is denied.

[EEXIST] O_CREAT and O_EXCL are set and the named shared memory object already exists.

[EINTR] The *shm_open*( ) operation was interrupted by a signal.

[EINVAL] The *shm_open*( ) operation is not supported for the given name.

[EMFILE] Too many file descriptors are currently in use by this process.

[ENAMETOOLONG] The length of the *name* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENFILE] Too many shared memory objects are currently open in the system.

[ENOENT] O_CREAT is not set and the named shared memory object does not exist.

[ENOSPC] There is insufficient space for the creation of the new shared memory object.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

When the Memory Mapped Files option is supported, the normal *open*( ) call is used to obtain a descriptor to a file to be mapped according to existing practice with *mmap*( ). When the Shared Memory Objects option is supported, the *shm_open*( ) function shall obtain a descriptor to the shared memory object to be mapped.

There is ample precedent for having a file descriptor represent several types of objects. In the POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory. Many implementations simply have an operations vector, which is indexed by the file descriptor type and does very different operations. Note that in some cases the file descriptor passed to generic operations on file descriptors is returned by *open*( ) or *creat*( ) and in some cases returned by alternate functions, such as *pipe*( ). The latter technique is used by *shm_open*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note that such shared memory objects can actually be implemented as mapped files. In both cases, the size can be set after the open using *ftruncate*(). The *shm_open*() function itself does not create a shared object of a specified size because this would duplicate an extant function that set the size of an object referenced by a file descriptor.

On implementations where memory objects are implemented using the existing file system, the *shm_open*() function may be implemented using a macro that invokes *open*(), and the *shm_unlink*() function may be implemented using a macro that invokes *unlink*().

For implementations without a permanent file system, the definition of the name of the memory objects is allowed not to survive a system reboot. Note that this allows systems with a permanent file system to implement memory objects as data structures internal to the implementation as well.

On implementations that choose to implement memory objects using memory directly, a *shm_open*() followed by an *ftruncate*() and *close*() can be used to preallocate a shared memory area and to set the size of that preallocation. This may be necessary for systems without virtual memory hardware support in order to ensure that the memory is contiguous.

The set of valid open flags to *shm_open*() was restricted to O_RDONLY, O_RDWR, O_CREAT, and O_TRUNC because these could be easily implemented on most memory mapping systems. This volume of IEEE Std 1003.1-2001 is silent on the results if the implementation cannot supply the requested file access because of implementation-defined reasons, including hardware ones.

The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the implementation cannot complete a request.

[EACCES] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode because it conflicts with another requested mode. An example might be that an application desires to open a memory object two times, mapping different areas with different access modes. If the implementation cannot map a single area into a process space in two places, which would be required if different access modes were required for the two areas, then the implementation may inform the application at the time of the second open.

[ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode at all. An example would be that the hardware of the implementation cannot support write-only shared memory areas.

On all implementations, it may be desirable to restrict the location of the memory objects to specific file systems for performance (such as a RAM disk) or implementation-defined reasons (shared memory supported directly only on certain file systems). The *shm_open*() function may be used to enforce these restrictions. There are a number of methods available to the application to determine an appropriate name of the file or the location of an appropriate directory. One way is from the environment via *getenv*(). Another would be from a configuration file.

This volume of IEEE Std 1003.1-2001 specifies that memory objects have initial contents of zero when created. This is consistent with current behavior for both files and newly allocated memory. For those implementations that use physical memory, it would be possible that such implementations could simply use available memory and give it to the process uninitialized. This, however, is not consistent with standard behavior for the uninitialized data area, the stack, and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security reasons. Thus, initializing memory objects to zero is required.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

41308 **FUTURE DIRECTIONS**

41309 None.

41310 **SEE ALSO**

41311 *close*( ), *dup*( ), *exec*, *fcntl*( ), *mmap*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_unlink*( ), *umask*( ), the Base
41312 Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<sys/mman.h>**

41313 **CHANGE HISTORY**

41314 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41315 **Issue 6**

41316 The *shm_open*( ) function is marked as part of the Shared Memory Objects option.

41317 The [ENOSYS] error condition has been removed as stubs need not be provided if an
41318 implementation does not support the Shared Memory Objects option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

41319 **NAME**

41320 shm_unlink — remove a shared memory object (**REALTIME**)

41321 **SYNOPSIS**

41322 SHM
```
#include <sys/mman.h>

int shm_unlink(const char *name);
```
41323

41324

41325 **DESCRIPTION**

41326 The *shm_unlink*( ) function shall remove the name of the shared memory object named by the
41327 string pointed to by *name*.

41328 If one or more references to the shared memory object exist when the object is unlinked, the
41329 name shall be removed before *shm_unlink*( ) returns, but the removal of the memory object
41330 contents shall be postponed until all open and map references to the shared memory object have
41331 been removed.

41332 Even if the object continues to exist after the last *shm_unlink*( ), reuse of the name shall
41333 subsequently cause *shm_open*( ) to behave as if no shared memory object of this name exists (that
41334 is, *shm_open*( ) will fail if O_CREAT is not set, or will create a new shared memory object if
41335 O_CREAT is set).

41336 **RETURN VALUE**

41337 Upon successful completion, a value of zero shall be returned. Otherwise, a value of –1 shall be
41338 returned and *errno* set to indicate the error. If –1 is returned, the named shared memory object
41339 shall not be changed by this function call.

41340 **ERRORS**

41341 The *shm_unlink*( ) function shall fail if:

41342 [EACCES] Permission is denied to unlink the named shared memory object.

41343 [ENAMETOOLONG]

41344 The length of the *name* argument exceeds {PATH_MAX} or a pathname
41345 component is longer than {NAME_MAX}.

41346 [ENOENT] The named shared memory object does not exist.

41347 **EXAMPLES**

41348 None.

41349 **APPLICATION USAGE**

41350 Names of memory objects that were allocated with *open*( ) are deleted with *unlink*( ) in the usual
41351 fashion. Names of memory objects that were allocated with *shm_open*( ) are deleted with
41352 *shm_unlink*( ). Note that the actual memory object is not destroyed until the last close and
41353 unmap on it have occurred if it was already in use.

41354 **RATIONALE**

41355 None.

41356 **FUTURE DIRECTIONS**

41357 None.

41358 **SEE ALSO**

41359 *close*( ), *mmap*( ), *munmap*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_open*( ), the Base Definitions volume
41360 of IEEE Std 1003.1-2001, **<sys/mman.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

41361 **CHANGE HISTORY**

41362 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41363 **Issue 6**

41364 The *shm_unlink*( ) function is marked as part of the Shared Memory Objects option.

41365 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm_unlink*( )
41366 will not attach to the old shared memory object.

41367 The [ENOSYS] error condition has been removed as stubs need not be provided if an
41368 implementation does not support the Shared Memory Objects option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

shmat — XSI shared memory attach operation

SYNOPSIS

XSI
```
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

DESCRIPTION

The *shmat*( ) function operates on XSI shared memory (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *shmat*( ) function attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the address space of the calling process. The segment is attached at the address specified by one of the following criteria:

- If *shmaddr* is a null pointer, the segment is attached at the first available address as selected by the system.
- If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is non-zero, the segment is attached at the address given by (*shmaddr* –((*uintptr_t*)*shmaddr* %SHMLBA)). The character '%' is the C-language remainder operator.
- If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is 0, the segment is attached at the address given by *shmaddr*.
- The segment is attached for reading if (*shmflg* &SHM_RDONLY) is non-zero and the calling process has read permission; otherwise, if it is 0 and the calling process has read and write permission, the segment is attached for reading and writing.

RETURN VALUE

Upon successful completion, *shmat*( ) shall increment the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return the segment's start address.

Otherwise, the shared memory segment shall not be attached, *shmat*( ) shall return –1, and *errno* shall be set to indicate the error.

ERRORS

The *shmat*( ) function shall fail if:

[EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page 39).

[EINVAL] The value of *shmid* is not a valid shared memory identifier, the *shmaddr* is not a null pointer, and the value of (*shmaddr* –((*uintptr_t*)*shmaddr* %SHMLBA)) is an illegal address for attaching shared memory; or the *shmaddr* is not a null pointer, (*shmflg* &SHM_RND) is 0, and the value of *shmaddr* is an illegal address for attaching shared memory.

[EMFILE] The number of shared memory segments attached to the calling process would exceed the system-imposed limit.

[ENOMEM] The available data space is not large enough to accommodate the shared memory segment.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit*( ), *fork*( ), *shmctl*( ), *shmdt*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/shm.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

Moved from SHARED MEMORY to BASE.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 6

The Open Group Corrigendum U021/13 is applied.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

shmctl — XSI shared memory control operations

SYNOPSIS

XSI
```
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

DESCRIPTION

The *shmctl*( ) function operates on XSI shared memory (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *shmctl*( ) function provides a variety of shared memory control operations as specified by *cmd*. The following values for *cmd* are available:

IPC_STAT — Place the current value of each member of the **shmid_ds** data structure associated with *shmid* into the structure pointed to by *buf*. The contents of the structure are defined in **<sys/shm.h>**.

IPC_SET — Set the value of the following members of the **shmid_ds** data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

`shm_perm.uid`
`shm_perm.gid`
`shm_perm.mode` Low-order nine bits.

IPC_SET can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shmid_ds** data structure associated with *shmid*.

IPC_RMID — Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and **shmid_ds** data structure associated with it. IPC_RMID can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shmid_ds** data structure associated with *shmid*.

RETURN VALUE

Upon successful completion, *shmctl*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *shmctl*( ) function shall fail if:

[EACCES] — The argument *cmd* is equal to IPC_STAT and the calling process does not have read permission; see Section 2.7 (on page 39).

[EINVAL] — The value of *shmid* is not a valid shared memory identifier, or the value of *cmd* is not a valid command.

[EPERM] — The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in the data structure associated with *shmid*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *shmctl*( ) function may fail if:

[EOVERFLOW] The *cmd* argument is IPC_STAT and the *gid* or *uid* value is too large to be stored in the structure pointed to by the *buf* argument.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *shmat*( ), *shmdt*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/shm.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

Moved from SHARED MEMORY to BASE.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

shmdt — XSI shared memory detach operation

SYNOPSIS

XSI
```
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

DESCRIPTION

The *shmdt*( ) function operates on XSI shared memory (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *shmdt*( ) function detaches the shared memory segment located at the address specified by *shmaddr* from the address space of the calling process.

RETURN VALUE

Upon successful completion, *shmdt*( ) shall decrement the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return 0.

Otherwise, the shared memory segment shall not be detached, *shmdt*( ) shall return –1, and *errno* shall be set to indicate the error.

ERRORS

The *shmdt*( ) function shall fail if:

[EINVAL] The value of *shmaddr* is not the data segment start address of a shared memory segment.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit*( ), *fork*( ), *shmat*( ), *shmctl*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/shm.h>**

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

Moved from SHARED MEMORY to BASE.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

shmget — get an XSI shared memory segment

SYNOPSIS

XSI
```
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
```

DESCRIPTION

The *shmget*( ) function operates on XSI shared memory (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 41).

The *shmget*( ) function shall return the shared memory identifier associated with *key*.

A shared memory identifier, associated data structure, and shared memory segment of at least *size* bytes (see **<sys/shm.h>**) are created for *key* if one of the following is true:

- The argument *key* is equal to IPC_PRIVATE.
- The argument *key* does not already have a shared memory identifier associated with it and (*shmflg* &IPC_CREAT) is non-zero.

Upon creation, the data structure associated with the new shared memory identifier shall be initialized as follows:

- The values of *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order nine bits of *shm_perm.mode* are set equal to the low-order nine bits of *shmflg*.
- The value of *shm_segsz* is set equal to the value of *size*.
- The values of *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set equal to 0.
- The value of *shm_ctime* is set equal to the current time.

When the shared memory segment is created, it shall be initialized with all zero values.

RETURN VALUE

Upon successful completion, *shmget*( ) shall return a non-negative integer, namely a shared memory identifier; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *shmget*( ) function shall fail if:

[EACCES] A shared memory identifier exists for *key* but operation permission as specified by the low-order nine bits of *shmflg* would not be granted; see Section 2.7 (on page 39).

[EEXIST] A shared memory identifier exists for the argument *key* but (*shmflg* &IPC_CREAT) &&(*shmflg* &IPC_EXCL) is non-zero.

[EINVAL] A shared memory segment is to be created and the value of size is less than the system-imposed minimum or greater than the system-imposed maximum.

[EINVAL] No shared memory segment is to be created and a shared memory segment exists for *key* but the size of the segment associated with it is less than *size* and *size* is not 0.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

41586 [ENOENT] A shared memory identifier does not exist for the argument *key* and (*shmflg*
41587 &IPC_CREAT) is 0.

41588 [ENOMEM] A shared memory identifier and associated shared memory segment shall be
41589 created, but the amount of available physical memory is not sufficient to fill
41590 the request.

41591 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on
41592 the maximum number of allowed shared memory identifiers system-wide
41593 would be exceeded.

41594 **EXAMPLES**

41595 None.

41596 **APPLICATION USAGE**

41597 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
41598 Application developers who need to use IPC should design their applications so that modules
41599 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
41600 alternative interfaces.

41601 **RATIONALE**

41602 None.

41603 **FUTURE DIRECTIONS**

41604 None.

41605 **SEE ALSO**

41606 Section 2.7 (on page 39), Section 2.8 (on page 41), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_open*( ),
41607 *shm_unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/shm.h>**

41608 **CHANGE HISTORY**

41609 First released in Issue 2. Derived from Issue 2 of the SVID.

41610 **Issue 5**

41611 Moved from SHARED MEMORY to BASE.

41612 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
41613 DIRECTIONS to a new APPLICATION USAGE section.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

41614 **NAME**

41615 shutdown — shut down socket send and receive operations

41616 **SYNOPSIS**

41617 `#include <sys/socket.h>`

41618 `int shutdown(int socket, int how);`

41619 **DESCRIPTION**

41620 The *shutdown*( ) function shall cause all or part of a full-duplex connection on the socket
41621 associated with the file descriptor *socket* to be shut down.

41622 The *shutdown*( ) function takes the following arguments:

41623 *socket* Specifies the file descriptor of the socket.

41624 *how* Specifies the type of shutdown. The values are as follows:

41625 SHUT_RD Disables further receive operations.

41626 SHUT_WR Disables further send operations.

41627 SHUT_RDWR Disables further send and receive operations.

41628 The *shutdown*( ) function disables subsequent send and/or receive operations on a socket,
41629 depending on the value of the *how* argument.

41630 **RETURN VALUE**

41631 Upon successful completion, *shutdown*( ) shall return 0; otherwise, –1 shall be returned and *errno*
41632 set to indicate the error.

41633 **ERRORS**

41634 The *shutdown*( ) function shall fail if:

41635 [EBADF] The *socket* argument is not a valid file descriptor.

41636 [EINVAL] The *how* argument is invalid.

41637 [ENOTCONN] The socket is not connected.

41638 [ENOTSOCK] The *socket* argument does not refer to a socket.

41639 The *shutdown*( ) function may fail if:

41640 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

41641 **EXAMPLES**

41642 None.

41643 **APPLICATION USAGE**

41644 None.

41645 **RATIONALE**

41646 None.

41647 **FUTURE DIRECTIONS**

41648 None.

41649 **SEE ALSO**

41650 *getsockopt*( ), *read*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *select*( ), *send*( ), *sendto*( ), *setsockopt*( ), *socket*( ),
41651 *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

41652 **CHANGE HISTORY**

41653 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sigaction — examine and change a signal action

SYNOPSIS

CX
```
#include <signal.h>

int sigaction(int sig, const struct sigaction *restrict act,
    struct sigaction *restrict oact);
```

DESCRIPTION

The *sigaction*( ) function allows the calling process to examine and/or specify the action to be associated with a specific signal. The argument *sig* specifies the signal; acceptable values are defined in **<signal.h>**.

The structure **sigaction**, used to describe an action to be taken, is defined in the **<signal.h>** header to include at least the following members:

Member Type	Member Name	Description
void(*) (int)	*sa_handler*	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
sigset_t	*sa_mask*	Additional set of signals to be blocked during execution of signal-catching function.
int	*sa_flags*	Special flags to affect behavior of signal.
void(*) (int, siginfo_t *, void *)	*sa_sigaction*	Pointer to a signal-catching function.

The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application shall not use both simultaneously.

If the argument *act* is not a null pointer, it points to a structure specifying the action to be associated with the specified signal. If the argument *oact* is not a null pointer, the action previously associated with the signal is stored in the location pointed to by the argument *oact*. If the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall not be added to the signal mask using this mechanism; this restriction shall be enforced by the system without causing an error to be indicated.

If the SA_SIGINFO flag (see below) is cleared in the *sa_flags* field of the **sigaction** structure, the XSI|RTS *sa_handler* field identifies the action to be associated with the specified signal. If the SA_SIGINFO flag is set in the *sa_flags* field, and the implementation supports the Realtime Signals Extension option or the XSI Extension option, the *sa_sigaction* field specifies a signal-catching function. If the SA_SIGINFO bit is cleared and the *sa_handler* field specifies a signal-catching function, or if the SA_SIGINFO bit is set, the *sa_mask* field identifies a set of signals that shall be added to the signal mask of the thread before the signal-catching function is invoked. If the *sa_handler* field specifies a signal-catching function, the *sa_mask* field identifies a set of signals that shall be added to the process' signal mask before the signal-catching function is invoked.

The *sa_flags* field can be used to modify the behavior of the specified signal.

The following flags, defined in the **<signal.h>** header, can be set in *sa_flags*:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

41699 XSI SA_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children
41700 continue.

41701 If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is not set in *sa_flags*, and
41702 the implementation supports the SIGCHLD signal, then a SIGCHLD
41703 signal shall be generated for the calling process whenever any of its child
41704 XSI processes stop and a SIGCHLD signal may be generated for the calling
41705 process whenever any of its stopped child processes are continued. If *sig*
41706 is SIGCHLD and the SA_NOCLDSTOP flag is set in *sa_flags*, then the
41707 implementation shall not generate a SIGCHLD signal in this way.

41708 XSI SA_ONSTACK If set and an alternate signal stack has been declared with *sigaltstack*(), the
41709 signal shall be delivered to the calling process on that stack. Otherwise,
41710 the signal shall be delivered on the current stack.

41711 XSI SA_RESETHAND If set, the disposition of the signal shall be reset to SIG_DFL and the
41712 SA_SIGINFO flag shall be cleared on entry to the signal handler.

41713 **Note:** SIGILL and SIGTRAP cannot be automatically reset when delivered;
41714 the system silently enforces this restriction.

41715 Otherwise, the disposition of the signal shall not be modified on entry to
41716 the signal handler.

41717 In addition, if this flag is set, *sigaction*() behaves as if the SA_NODEFER
41718 flag were also set.

41719 XSI SA_RESTART This flag affects the behavior of interruptible functions; that is, those
41720 specified to fail with *errno* set to [EINTR]. If set, and a function specified
41721 as interruptible is interrupted by this signal, the function shall restart and
41722 shall not fail with [EINTR] unless otherwise specified. If the flag is not
41723 set, interruptible functions interrupted by this signal shall fail with *errno*
41724 set to [EINTR].

41725 SA_SIGINFO If cleared and the signal is caught, the signal-catching function shall be
41726 entered as:

41727 `void func(int signo);`

41728 where *signo* is the only argument to the signal-catching function. In this
41729 case, the application shall use the *sa_handler* member to describe the
41730 signal-catching function and the application shall not modify the
41731 *sa_sigaction* member.

41732 XSI|RTS If SA_SIGINFO is set and the signal is caught, the signal-catching
41733 function shall be entered as:

41734 `void func(int signo, siginfo_t *info, void *context);`

41735 where two additional arguments are passed to the signal-catching
41736 function. The second argument shall point to an object of type **siginfo_t**
41737 explaining the reason why the signal was generated; the third argument
41738 can be cast to a pointer to an object of type **ucontext_t** to refer to the
41739 receiving process' context that was interrupted when the signal was
41740 delivered. In this case, the application shall use the *sa_sigaction* member
41741 to describe the signal-catching function and the application shall not
41742 modify the *sa_handler* member.

41743 The *si_signo* member contains the system-generated signal number.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

41744 XSI The *si_errno* member may contain implementation-defined additional
41745 error information; if non-zero, it contains an error number identifying the
41746 condition that caused the signal to be generated.

41747 XSI | RTS The *si_code* member contains a code identifying the cause of the signal.

41748 XSI If the value of *si_code* is less than or equal to 0, then the signal was
41749 generated by a process and *si_pid* and *si_uid*, respectively, indicate the
41750 process ID and the real user ID of the sender. The **<signal.h>** header
41751 description contains information about the signal-specific contents of the
41752 elements of the **siginfo_t** type.

41753 XSI SA_NOCLDWAIT If set, and *sig* equals SIGCHLD, child processes of the calling processes
41754 shall not be transformed into zombie processes when they terminate. If
41755 the calling process subsequently waits for its children, and the process
41756 has no unwaited-for children that were transformed into zombie
41757 processes, it shall block until all of its children terminate, and *wait*(),
41758 *waitid*(), and *waitpid*() shall fail and set *errno* to [ECHILD]. Otherwise,
41759 terminating child processes shall be transformed into zombie processes,
41760 unless SIGCHLD is set to SIG_IGN.

41761 XSI SA_NODEFER If set and *sig* is caught, *sig* shall not be added to the process' signal mask
41762 on entry to the signal handler unless it is included in *sa_mask*. Otherwise,
41763 *sig* shall always be added to the process' signal mask on entry to the
41764 signal handler.

41765 When a signal is caught by a signal-catching function installed by *sigaction*(), a new signal mask
41766 is calculated and installed for the duration of the signal-catching function (or until a call to either
41767 *sigprocmask*() or *sigsuspend*() is made). This mask is formed by taking the union of the current
41768 XSI signal mask and the value of the *sa_mask* for the signal being delivered unless SA_NODEFER or
41769 SA_RESETHAND is set, and then including the signal being delivered. If and when the user's
41770 signal handler returns normally, the original signal mask is restored.

41771 Once an action is installed for a specific signal, it shall remain installed until another action is
41772 XSI explicitly requested (by another call to *sigaction*()), until the SA_RESETHAND flag causes
41773 resetting of the handler, or until one of the *exec* functions is called.

41774 If the previous action for *sig* had been established by *signal*(), the values of the fields returned in
41775 the structure pointed to by *oact* are unspecified, and in particular *oact->sa_handler* is not
41776 necessarily the same value passed to *signal*(). However, if a pointer to the same structure or a
41777 copy thereof is passed to a subsequent call to *sigaction*() via the *act* argument, handling of the
41778 signal shall be as if the original call to *signal*() were repeated.

41779 If *sigaction*() fails, no new signal handler is installed.

41780 It is unspecified whether an attempt to set the action for a signal that cannot be caught or
41781 ignored to SIG_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

41782 If SA_SIGINFO is not set in *sa_flags*, then the disposition of subsequent occurrences of *sig* when
41783 it is already pending is implementation-defined; the signal-catching function shall be invoked
41784 RTS with a single argument. If the implementation supports the Realtime Signals Extension option,
41785 and if SA_SIGINFO is set in *sa_flags*, then subsequent occurrences of *sig* generated by *sigqueue*()
41786 or as a result of any signal-generating function that supports the specification of an application-
41787 defined value (when *sig* is already pending) shall be queued in FIFO order until delivered or
41788 accepted; the signal-catching function shall be invoked with three arguments. The application
41789 specified value is passed to the signal-catching function as the *si_value* member of the **siginfo_t**
41790 structure.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The result of the use of *sigaction*( ) and a *sigwait*( ) function concurrently within a process on the same signal is unspecified.

RETURN VALUE

Upon successful completion, *sigaction*( ) shall return 0; otherwise, –1 shall be returned, *errno* shall be set to indicate the error, and no new signal-catching function shall be installed.

ERRORS

The *sigaction*( ) function shall fail if:

[EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

[ENOTSUP] The SA_SIGINFO bit flag is set in the *sa_flags* field of the **sigaction** structure, and the implementation does not support either the Realtime Signals Extension option, or the XSI Extension option.

The *sigaction*( ) function may fail if:

[EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).

EXAMPLES

None.

APPLICATION USAGE

The *sigaction*( ) function supersedes the *signal*( ) function, and should be used in preference. In particular, *sigaction*( ) and *signal*( ) should not be used in the same process to control the same signal. The behavior of reentrant functions, as defined in the DESCRIPTION, is as specified by this volume of IEEE Std 1003.1-2001, regardless of invocation from a signal-catching function. This is the only intended meaning of the statement that reentrant functions may be used in signal-catching functions without restrictions. Applications must still consider all effects of such functions on such things as data structures, files, and process state. In particular, application writers need to consider the restrictions on interactions when interrupting *sleep*( ) and interactions among multiple handles for a file description. The fact that any specific function is listed as reentrant does not necessarily mean that invocation of that function from a signal-catching function is recommended.

In order to prevent errors arising from interrupting non-reentrant function calls, applications should protect calls to these functions either by blocking the appropriate signals or through the use of some programmatic semaphore (see *semget*( ), *sem_init*( ), *sem_open*( ), and so on). Note in particular that even the ''safe'' functions may modify *errno*; the signal-catching function, if not executing as an independent thread, may want to save and restore its value. Naturally, the same principles apply to the reentrancy of application routines and asynchronous data access. Note that *longjmp*( ) and *siglongjmp*( ) are not in the list of reentrant functions. This is because the code executing after *longjmp*( ) and *siglongjmp*( ) can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use *longjmp*( ) and *siglongjmp*( ) from within signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either *malloc*( ) or *free*( ) functions or the standard I/O library, both of which traditionally use data structures in a non-reentrant manner. Since any combination of different functions using a common data structure can cause reentrancy problems, this volume of IEEE Std 1003.1-2001 does not define the behavior when any unsafe function is called in a signal handler that interrupts an unsafe function.

If the signal occurs other than as the result of calling *abort*( ), *kill*( ), or *raise*( ), the behavior is undefined if the signal handler calls any function in the standard library other than one of the

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

functions listed in the table above or refers to any object with static storage duration other than by assigning a value to a static storage duration variable of type **volatile sig_atomic_t**. Furthermore, if such a call fails, the value of *errno* is unspecified.

Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving process resumes execution at the point it was interrupted unless the signal handler makes other arrangements. If *longjmp*( ) or *_longjmp*( ) is used to leave the signal handler, then the signal mask must be explicitly restored by the process.

This volume of IEEE Std 1003.1-2001 defines the third argument of a signal handling function when SA_SIGINFO is set as a **void *** instead of a **ucontext_t ***, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to **ucontext_t ***.

The BSD optional four argument signal handling function is not supported by this volume of IEEE Std 1003.1-2001. The BSD declaration would be:

```
void handler(int sig, int code, struct sigcontext *scp,
    char *addr);
```

where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer to the **sigcontext** structure, and *addr* is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when SA_SIGINFO is set.

RATIONALE

Although this volume of IEEE Std 1003.1-2001 requires that signals that cannot be ignored shall not be added to the signal mask when a signal-catching function is entered, there is no explicit requirement that subsequent calls to *sigaction*( ) reflect this in the information returned in the *oact* argument. In other words, if SIGKILL is included in the *sa_mask* field of *act*, it is unspecified whether or not a subsequent call to *sigaction*( ) returns with SIGKILL included in the *sa_mask* field of *oact*.

The SA_NOCLDSTOP flag, when supplied in the *act->sa_flags* parameter, allows overloading SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated child. Most conforming applications that catch SIGCHLD are expected to install signal-catching functions that repeatedly call the *waitpid*( ) function with the WNOHANG flag set, acting on each child for which status is returned, until *waitpid*( ) returns zero. If stopped children are not of interest, the use of the SA_NOCLDSTOP flag can prevent the overhead from invoking the signal-catching routine when they stop.

Some historical implementations also define other mechanisms for stopping processes, such as the *ptrace*( ) function. These implementations usually do not generate a SIGCHLD signal when processes stop due to this mechanism; however, that is beyond the scope of this volume of IEEE Std 1003.1-2001.

This volume of IEEE Std 1003.1-2001 requires that calls to *sigaction*( ) that supply a NULL *act* argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or SIGSTOP). The System V *signal*( ) and BSD *sigvec*( ) functions return [EINVAL] in these cases and, in this respect, their behavior varies from *sigaction*( ).

This volume of IEEE Std 1003.1-2001 requires that *sigaction*( ) properly save and restore a signal action set up by the ISO C standard *signal*( ) function. However, there is no guarantee that the reverse is true, nor could there be given the greater amount of information conveyed by the **sigaction** structure. Because of this, applications should avoid using both functions for the same signal in the same process. Since this cannot always be avoided in case of general-purpose

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

library routines, they should always be implemented with *sigaction*( ).

It was intended that the *signal*( ) function should be implementable as a library routine using *sigaction*( ).

The POSIX Realtime Extension extends the *sigaction*( ) function as specified by the POSIX.1-1990 standard to allow the application to request on a per-signal basis via an additional signal action flag that the extra parameters, including the application-defined signal value, if any, be passed to the signal-catching function.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 28), *bsd_signal*( ), *kill*( ), *_longjmp*( ), *longjmp*( ), *raise*( ), *semget*( ), *sem_init*( ), *sem_open*( ), *sigaddset*( ), *sigaltstack*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *signal*( ), *sigprocmask*( ), *sigsuspend*( ), *wait*( ), *waitid*( ), *waitpid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<ucontext.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX Threads Extension.

In the DESCRIPTION, the second argument to *func* when SA_SIGINFO is set is no longer permitted to be NULL, and the description of permitted **siginfo_t** contents is expanded by reference to **<signal.h>**.

Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP] error is deleted.

Issue 6

The Open Group Corrigendum U028/7 is applied. In the paragraph entitled ‘‘Signal Effects on Other Functions’’, a reference to *sigpending*( ) is added.

In the DESCRIPTION, the text ‘‘Signal Generation and Delivery’’, ‘‘Signal Actions’’, and ‘‘Signal Effects on Other Functions’’ are moved to a separate section of this volume of IEEE Std 1003.1-2001.

Text describing functionality from the Realtime Signals option is marked.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The [ENOTSUP] error condition is added.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *sigaction*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

References to the *wait3*( ) function are removed.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table describing the **sigaction** structure.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

41926 **NAME**

41927 sigaddset — add a signal to a signal set

41928 **SYNOPSIS**

41929 CX
```
#include <signal.h>

int sigaddset(sigset_t *set, int signo);
```
41930

41931

41932 **DESCRIPTION**

41933 The *sigaddset*( ) function adds the individual signal specified by the *signo* to the signal set pointed
41934 to by *set*.

41935 Applications shall call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type
41936 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
41937 nonetheless supplied as an argument to any of *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ),
41938 *sigdelset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), *sigtimedwait*( ), *sigwait*( ), or
41939 *sigwaitinfo*( ), the results are undefined.

41940 **RETURN VALUE**

41941 Upon successful completion, *sigaddset*( ) shall return 0; otherwise, it shall return –1 and set *errno*
41942 to indicate the error.

41943 **ERRORS**

41944 The *sigaddset*( ) function may fail if:

41945 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

41946 **EXAMPLES**

41947 None.

41948 **APPLICATION USAGE**

41949 None.

41950 **RATIONALE**

41951 None.

41952 **FUTURE DIRECTIONS**

41953 None.

41954 **SEE ALSO**

41955 Section 2.4 (on page 28), *sigaction*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ),
41956 *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
41957 **<signal.h>**

41958 **CHANGE HISTORY**

41959 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

41960 **Issue 5**

41961 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
41962 previous issues.

41963 **Issue 6**

41964 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

41965 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
41966 extension over the ISO C standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

41967 **NAME**

41968 sigaltstack — set and get signal alternate stack context

41969 **SYNOPSIS**

41970 XSI
```
#include <signal.h>

int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```
41971

41972

41973 **DESCRIPTION**

41974 The *sigaltstack*( ) function allows a process to define and examine the state of an alternate stack
41975 for signal handlers for the current thread. Signals that have been explicitly declared to execute
41976 on the alternate stack shall be delivered on the alternate stack.

41977 If *ss* is not a null pointer, it points to a **stack_t** structure that specifies the alternate signal stack
41978 that shall take effect upon return from *sigaltstack*( ). The *ss_flags* member specifies the new stack
41979 state. If it is set to SS_DISABLE, the stack is disabled and *ss_sp* and *ss_size* are ignored.
41980 Otherwise, the stack shall be enabled, and the *ss_sp* and *ss_size* members specify the new address
41981 and size of the stack.

41982 The range of addresses starting at *ss_sp* up to but not including *ss_sp*+*ss_size* is available to the
41983 implementation for use as the stack. This function makes no assumptions regarding which end
41984 is the stack base and in which direction the stack grows as items are pushed.

41985 If *oss* is not a null pointer, on successful completion it shall point to a **stack_t** structure that
41986 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack*( ). The *ss_sp*
41987 and *ss_size* members specify the address and size of that stack. The *ss_flags* member specifies the
41988 stack's state, and may contain one of the following values:

41989 SS_ONSTACK The process is currently executing on the alternate signal stack. Attempts to
41990 modify the alternate signal stack while the process is executing on it fail. This
41991 flag shall not be modified by processes.

41992 SS_DISABLE The alternate signal stack is currently disabled.

41993 The value SIGSTKSZ is a system default specifying the number of bytes that would be used to
41994 cover the usual case when manually allocating an alternate stack area. The value MINSIGSTKSZ
41995 is defined to be the minimum stack size for a signal handler. In computing an alternate stack
41996 size, a program should add that amount to its stack requirements to allow for the system
41997 implementation overhead. The constants SS_ONSTACK, SS_DISABLE, SIGSTKSZ, and
41998 MINSIGSTKSZ are defined in **<signal.h>**.

41999 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new
42000 process image.

42001 In some implementations, a signal (whether or not indicated to execute on the alternate stack)
42002 shall always execute on the alternate stack if it is delivered while another signal is being caught
42003 using the alternate stack.

42004 Use of this function by library threads that are not bound to kernel-scheduled entities results in
42005 undefined behavior.

42006 **RETURN VALUE**

42007 Upon successful completion, *sigaltstack*( ) shall return 0; otherwise, it shall return –1 and set *errno*
42008 to indicate the error.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

42009 **ERRORS**

42010 The *sigaltstack*( ) function shall fail if:

42011 [EINVAL] The *ss* argument is not a null pointer, and the *ss_flags* member pointed to by *ss*
42012 contains flags other than SS_DISABLE.

42013 [ENOMEM] The size of the alternate stack area is less than MINSIGSTKSZ.

42014 [EPERM] An attempt was made to modify an active stack.

42015 **EXAMPLES**

42016 **Allocating Memory for an Alternate Stack**

42017 The following example illustrates a method for allocating memory for an alternate stack.

```
42018 #include <signal.h>
42019 ...
42020 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
42021     /* Error return. */
42022 sigstk.ss_size = SIGSTKSZ;
42023 sigstk.ss_flags = 0;
42024 if (sigaltstack(&sigstk,(stack_t *)0) < 0)
42025     perror("sigaltstack");
```

42026 **APPLICATION USAGE**

42027 On some implementations, stack space is automatically extended as needed. On those
42028 implementations, automatic extension is typically not available for an alternate stack. If the stack
42029 overflows, the behavior is undefined.

42030 **RATIONALE**

42031 None.

42032 **FUTURE DIRECTIONS**

42033 None.

42034 **SEE ALSO**

42035 Section 2.4 (on page 28), *sigaction*( ), *sigsetjmp*( ), the Base Definitions volume of
42036 IEEE Std 1003.1-2001, **<signal.h>**

42037 **CHANGE HISTORY**

42038 First released in Issue 4, Version 2.

42039 **Issue 5**

42040 Moved from X/OPEN UNIX extension to BASE.

42041 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in
42042 previous issues.

42043 **Issue 6**

42044 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

42045 The **restrict** keyword is added to the *sigaltstack*( ) prototype for alignment with the
42046 ISO/IEC 9899:1999 standard.

42047 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence
42048 to include ‘‘for the current thread’’.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sigdelset — delete a signal from a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigdelset(sigset_t *set, int signo);
```

DESCRIPTION

The *sigdelset*( ) function deletes the individual signal specified by *signo* from the signal set pointed to by *set*.

Applications should call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), *sigtimedwait*( ), *sigwait*( ), or *sigwaitinfo*( ), the results are undefined.

RETURN VALUE

Upon successful completion, *sigdelset*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *sigdelset*( ) function may fail if:

[EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 28), *sigaction*( ), *sigaddset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sigemptyset — initialize and empty a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

DESCRIPTION

The *sigemptyset*( ) function initializes the signal set pointed to by *set*, such that all signals defined in IEEE Std 1003.1-2001 are excluded.

RETURN VALUE

Upon successful completion, *sigemptyset*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The implementation of the *sigemptyset*( ) (or *sigfillset*( )) function could quite trivially clear (or set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the structure, such as a version field, to permit binary-compatibility between releases where the size of the set varies. For such reasons, either *sigemptyset*( ) or *sigfillset*( ) must be called prior to any other use of the signal set, even if such use is read-only (for example, as an argument to *sigpending*( )). This function is not intended for dynamic allocation.

The *sigfillset*( ) and *sigemptyset*( ) functions require that the resulting signal set include (or exclude) all the signals defined in this volume of IEEE Std 1003.1-2001. Although it is outside the scope of this volume of IEEE Std 1003.1-2001 to place this requirement on signals that are implemented as extensions, it is recommended that implementation-defined signals also be affected by these functions. However, there may be a good reason for a particular signal not to be affected. For example, blocking or ignoring an implementation-defined signal may have undesirable side effects, whereas the default action for that signal is harmless. In such a case, it would be preferable for such a signal to be excluded from the signal set returned by *sigfillset*( ).

In early proposals there was no distinction between invalid and unsupported signals (the names of optional signals that were not supported by an implementation were not defined by that implementation). The [EINVAL] error was thus specified as a required error for invalid signals. With that distinction, it is not necessary to require implementations of these functions to determine whether an optional signal is actually supported, as that could have a significant performance impact for little value. The error could have been required for invalid signals and optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is optional in both cases.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

42133 **SEE ALSO**

42134 Section 2.4 (on page 28), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ),
42135 *sigprocmask*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

42136 **CHANGE HISTORY**

42137 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42138 **Issue 6**

42139 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
42140 extension over the ISO C standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sigfillset — initialize and fill a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigfillset(sigset_t *set);
```

DESCRIPTION

The *sigfillset*( ) function shall initialize the signal set pointed to by *set*, such that all signals defined in this volume of IEEE Std 1003.1-2001 are included.

RETURN VALUE

Upon successful completion, *sigfillset*( ) shall return 0; otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *sigemptyset*( ) (on page 1354).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 28), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sighold, sigignore, sigpause, sigrelse, sigset — signal management

SYNOPSIS

XSI
```
#include <signal.h>

int sighold(int sig);
int sigignore(int sig);
int sigpause(int sig);
int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```

DESCRIPTION

Use of any of these functions is unspecified in a multi-threaded process.

The *sighold*( ), *sigignore*( ), *sigpause*( ), *sigrelse*( ), and *sigset*( ) functions provide simplified signal management.

The *sigset*( ) function shall modify signal dispositions. The *sig* argument specifies the signal, which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's disposition, which may be SIG_DFL, SIG_IGN, or the address of a signal handler. If *sigset*( ) is used, and *disp* is the address of a signal handler, the system shall add *sig* to the calling process' signal mask before executing the signal handler; when the signal handler returns, the system shall restore the calling process' signal mask to its state prior to the delivery of the signal. In addition, if *sigset*( ) is used, and *disp* is equal to SIG_HOLD, *sig* shall be added to the calling process' signal mask and *sig*'s disposition shall remain unchanged. If *sigset*( ) is used, and *disp* is not equal to SIG_HOLD, *sig* shall be removed from the calling process' signal mask.

The *sighold*( ) function shall add *sig* to the calling process' signal mask.

The *sigrelse*( ) function shall remove *sig* from the calling process' signal mask.

The *sigignore*( ) function shall set the disposition of *sig* to SIG_IGN.

The *sigpause*( ) function shall remove *sig* from the calling process' signal mask and suspend the calling process until a signal is received. The *sigpause*( ) function shall restore the process' signal mask to its original state before returning.

If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and *wait*( ), *waitid*( ), and *waitpid*( ) shall fail and set *errno* to [ECHILD].

RETURN VALUE

Upon successful completion, *sigset*( ) shall return SIG_HOLD if the signal had been blocked and the signal's previous disposition if it had not been blocked. Otherwise, SIG_ERR shall be returned and *errno* set to indicate the error.

The *sigpause*( ) function shall suspend execution of the thread until a signal is received, whereupon it shall return –1 and set *errno* to [EINTR].

For all other functions, upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

42214 **ERRORS**

42215 These functions shall fail if:

42216 [EINVAL] The *sig* argument is an illegal signal number.

42217 The *sigset*( ) and *sigignore*( ) functions shall fail if:

42218 [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a
42219 signal that cannot be ignored.

42220 **EXAMPLES**

42221 None.

42222 **APPLICATION USAGE**

42223 The *sigaction*( ) function provides a more comprehensive and reliable mechanism for controlling
42224 signals; new applications should use *sigaction*( ) rather than *sigset*( ).

42225 The *sighold*( ) function, in conjunction with *sigrelse*( ) or *sigpause*( ), may be used to establish
42226 critical regions of code that require the delivery of a signal to be temporarily deferred.

42227 The *sigsuspend*( ) function should be used in preference to *sigpause*( ) for broader portability.

42228 **RATIONALE**

42229 None.

42230 **FUTURE DIRECTIONS**

42231 None.

42232 **SEE ALSO**

42233 Section 2.4 (on page 28), *exec*, *pause*( ), *sigaction*( ), *signal*( ), *sigsuspend*( ), *waitid*( ), the Base
42234 Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

42235 **CHANGE HISTORY**

42236 First released in Issue 4, Version 2.

42237 **Issue 5**

42238 Moved from X/OPEN UNIX extension to BASE.

42239 The DESCRIPTION is updated to indicate that the *sigpause*( ) function restores the process'
42240 signal mask to its original state before returning.

42241 The RETURN VALUE section is updated to indicate that the *sigpause*( ) function suspends
42242 execution of the process until a signal is received, whereupon it returns –1 and sets *errno* to
42243 [EINTR].

42244 **Issue 6**

42245 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

42246 References to the *wait3*( ) function are removed.

42247 The XSI functions are split out into their own reference page.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

siginterrupt — allow signals to interrupt functions

SYNOPSIS

XSI

```
#include <signal.h>

int siginterrupt(int sig, int flag);
```

DESCRIPTION

The *siginterrupt*( ) function shall change the restart behavior when a function is interrupted by the specified signal. The function *siginterrupt*(*sig*, *flag*) has an effect as if implemented as:

```
int siginterrupt(int sig, int flag) {
    int ret;
    struct sigaction act;

    (void) sigaction(sig, NULL, &act);
    if (flag)
        act.sa_flags &= ~SA_RESTART;
    else
        act.sa_flags |= SA_RESTART;
    ret = sigaction(sig, &act, NULL);
    return ret;
}
```

RETURN VALUE

Upon successful completion, *siginterrupt*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *siginterrupt*( ) function shall fail if:

[EINVAL] The *sig* argument is not a valid signal number.

EXAMPLES

None.

APPLICATION USAGE

The *siginterrupt*( ) function supports programs written to historical system interfaces. A conforming application, when being written or rewritten, should use *sigaction*( ) with the SA_RESTART flag instead of *siginterrupt*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 28), *sigaction*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/59 is applied, correcting the declaration in the sample implementation given in the DESCRIPTION.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

sigismember — test for a signal in a signal set

SYNOPSIS

CX
```
#include <signal.h>

int sigismember(const sigset_t *set, int signo);
```

DESCRIPTION

The *sigismember*( ) function shall test whether the signal specified by *signo* is a member of the set pointed to by *set*.

Applications should call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), *sigtimedwait*( ), *sigwait*( ), or *sigwaitinfo*( ), the results are undefined.

RETURN VALUE

Upon successful completion, *sigismember*( ) shall return 1 if the specified signal is a member of the specified set, or 0 if it is not. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

The *sigismember*( ) function may fail if:

[EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 28), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigfillset*( ), *sigemptyset*( ), *sigpending*( ), *sigprocmask*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

siglongjmp — non-local goto with signal handling

SYNOPSIS

CX
```
#include <setjmp.h>

void siglongjmp(sigjmp_buf env, int val);
```

DESCRIPTION

The *siglongjmp*( ) function shall be equivalent to the *longjmp*( ) function, except as follows:

- References to *setjmp*( ) shall be equivalent to *sigsetjmp*( ).
- The *siglongjmp*( ) function shall restore the saved signal mask if and only if the *env* argument was initialized by a call to *sigsetjmp*( ) with a non-zero *savemask* argument.

RETURN VALUE

After *siglongjmp*( ) is completed, program execution shall continue as if the corresponding invocation of *sigsetjmp*( ) had just returned the value specified by *val*. The *siglongjmp*( ) function shall not cause *sigsetjmp*( ) to return 0; if *val* is 0, *sigsetjmp*( ) shall return the value 1.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The distinction between *setjmp*( ) or *longjmp*( ) and *sigsetjmp*( ) or *siglongjmp*( ) is only significant for programs which use *sigaction*( ), *sigprocmask*( ), or *sigsuspend*( ).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

longjmp( ), *setjmp*( ), *sigprocmask*( ), *sigsetjmp*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<setjmp.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The DESCRIPTION is rewritten in terms of *longjmp*( ).

The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

signal — signal management

SYNOPSIS

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

CX Use of this function is unspecified in a multi-threaded process.

The *signal*( ) function chooses one of three ways in which receipt of the signal number *sig* is to be subsequently handled. If the value of *func* is SIG_DFL, default handling for that signal shall occur. If the value of *func* is SIG_IGN, the signal shall be ignored. Otherwise, the application shall ensure that *func* points to a function to be called when that signal occurs. An invocation of such a function because of a signal, or (recursively) of any further functions called by that invocation (other than functions in the standard library), is called a ''signal handler''.

When a signal occurs, and *func* points to a function, it is implementation-defined whether the equivalent of a:

```
signal(sig, SIG_DFL);
```

is executed or the implementation prevents some implementation-defined set of signals (at least including *sig*) from occurring until the current signal handling has completed. (If the value of *sig* is SIGILL, the implementation may alternatively define that no action is taken.) Next the equivalent of:

```
(*func)(sig);
```

is executed. If and when the function returns, if the value of *sig* was SIGFPE, SIGILL, or SIGSEGV or any other implementation-defined value corresponding to a computational exception, the behavior is undefined. Otherwise, the program shall resume execution at the point it was interrupted. If the signal occurs as the result of calling the *abort*( ), *raise*( ), *kill*( ), *pthread_kill*( ), or *sigqueue*( ) function, the signal handler shall not call the *raise*( ) function.

If the signal occurs other than as the result of calling *abort*( ), *raise*( ), *kill*( ), *pthread_kill*( ), or *sigqueue*( ), the behavior is undefined if the signal handler refers to any object with static storage duration other than by assigning a value to an object declared as volatile **sig_atomic_t**, or if the signal handler calls any function in the standard library other than one of the functions listed in Section 2.4 (on page 28). Furthermore, if such a call fails, the value of *errno* is unspecified.

At program start-up, the equivalent of:

```
signal(sig, SIG_IGN);
```

is executed for some signals, and the equivalent of:

```
signal(sig, SIG_DFL);
```

is executed for all other signals (see *exec*).

RETURN VALUE

If the request can be honored, *signal*( ) shall return the value of *func* for the most recent call to *signal*( ) for the specified signal *sig*. Otherwise, SIG_ERR shall be returned and a positive value shall be stored in *errno*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

42414 **ERRORS**

42415 The *signal*( ) function shall fail if:

42416 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
42417 signal that cannot be caught or ignore a signal that cannot be ignored.

42418 The *signal*( ) function may fail if:

42419 CX [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
42420 caught or ignored (or both).

42421 **EXAMPLES**

42422 None.

42423 **APPLICATION USAGE**

42424 The *sigaction*( ) function provides a more comprehensive and reliable mechanism for controlling
42425 signals; new applications should use *sigaction*( ) rather than *signal*( ).

42426 **RATIONALE**

42427 None.

42428 **FUTURE DIRECTIONS**

42429 None.

42430 **SEE ALSO**

42431 Section 2.4 (on page 28), *exec*, *pause*( ), *sigaction*( ), *sigsuspend*( ), *waitid*( ), the Base Definitions
42432 volume of IEEE Std 1003.1-2001, **<signal.h>**

42433 **CHANGE HISTORY**

42434 First released in Issue 1. Derived from Issue 1 of the SVID.

42435 **Issue 5**

42436 Moved from X/OPEN UNIX extension to BASE.

42437 The DESCRIPTION is updated to indicate that the *sigpause*( ) function restores the process'
42438 signal mask to its original state before returning.

42439 The RETURN VALUE section is updated to indicate that the *sigpause*( ) function suspends
42440 execution of the process until a signal is received, whereupon it returns –1 and sets *errno* to
42441 [EINTR].

42442 **Issue 6**

42443 Extensions beyond the ISO C standard are marked.

42444 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

42445 The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

42446 References to the *wait3*( ) function are removed.

42447 The *sighold*( ), *sigignore*( ), *sigrelse*( ), and *sigset*( ) functions are split out onto their own reference
42448 page.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

signbit — test sign

SYNOPSIS

```
#include <math.h>

int signbit(real-floating x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *signbit*( ) macro shall determine whether the sign of its argument value is negative. NaNs, zeros, and infinities have a sign bit.

RETURN VALUE

The *signbit*( ) macro shall return a non-zero value if and only if the sign of its argument value is negative.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isinf*( ), *isnan*( ), *isnormal*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

42478 **NAME**

42479 sigpause — remove a signal from the signal mask and suspend the thread

42480 **SYNOPSIS**

42481 XSI
```
#include <signal.h>
```

42482
```
int sigpause(int sig);
```

42483

42484 **DESCRIPTION**

42485 Refer to *sighold*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sigpending — examine pending signals

SYNOPSIS

CX
```
#include <signal.h>

int sigpending(sigset_t *set);
```

DESCRIPTION

The *sigpending*( ) function shall store, in the location referenced by the *set* argument, the set of signals that are blocked from delivery to the calling thread and that are pending on the process or the calling thread.

RETURN VALUE

Upon successful completion, *sigpending*( ) shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sigaddset( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigprocmask*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sigprocmask — examine and change blocked signals

SYNOPSIS

CX
```
#include <signal.h>

int sigprocmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
```

DESCRIPTION

Refer to *pthread_sigmask*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sigqueue — queue a signal to a process (**REALTIME**)

SYNOPSIS

RTS
```
#include <signal.h>

int sigqueue(pid_t pid, int signo, const union sigval value);
```

DESCRIPTION

The *sigqueue*( ) function shall cause the signal specified by *signo* to be sent with the value specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of *pid*.

The conditions required for a process to have permission to queue a signal to another process are the same as for the *kill*( ) function.

The *sigqueue*( ) function shall return immediately. If SA_SIGINFO is set for *signo* and if the resources were available to queue the signal, the signal shall be queued and sent to the receiving process. If SA_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving process; it is unspecified whether *value* shall be sent to the receiving process as a result of this call.

If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait*( ) function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the calling thread before the *sigqueue*( ) function returns. Should any multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.

RETURN VALUE

Upon successful completion, the specified signal shall have been queued, and the *sigqueue*( ) function shall return a value of zero. Otherwise, the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sigqueue*( ) function shall fail if:

[EAGAIN] No resources are available to queue the signal. The process has already queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or a system-wide resource limit has been exceeded.

[EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

[EPERM] The process does not have the appropriate privilege to send the signal to the receiving process.

[ESRCH] The process *pid* does not exist.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *sigqueue*( ) function allows an application to queue a realtime signal to itself or to another process, specifying the application-defined value. This is common practice in realtime applications on existing realtime systems. It was felt that specifying another function in the *sig*... name space already carved out for signals was preferable to extending the interface to *kill*( ).

Such a function became necessary when the put/get event function of the message queues was removed. It should be noted that the *sigqueue*( ) function implies reduced performance in a security-conscious implementation as the access permissions between the sender and receiver have to be checked on each send when the *pid* is resolved into a target process. Such access checks were necessary only at message queue open in the previous interface.

The standard developers required that *sigqueue*( ) have the same semantics with respect to the null signal as *kill*( ), and that the same permission checking be used. But because of the difficulty of implementing the ''broadcast'' semantic of *kill*( ) (for example, to process groups) and the interaction with resource allocation, this semantic was not adopted. The *sigqueue*( ) function queues a signal to a single process specified by the *pid* argument.

The *sigqueue*( ) function can fail if the system has insufficient resources to queue the signal. An explicit limit on the number of queued signals that a process could send was introduced. While the limit is ''per-sender'', this volume of IEEE Std 1003.1-2001 does not specify that the resources be part of the state of the sender. This would require either that the sender be maintained after exit until all signals that it had sent to other processes were handled or that all such signals that had not yet been acted upon be removed from the queue(s) of the receivers. This volume of IEEE Std 1003.1-2001 does not preclude this behavior, but an implementation that allocated queuing resources from a system-wide pool (with per-sender limits) and that leaves queued signals pending after the sender exits is also permitted.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.8.1 (on page 41), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The *sigqueue*( ) function is marked as part of the Realtime Signals Extension option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Realtime Signals Extension option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

42606 **NAME**

42607 sigrelse, sigset — signal management

42608 **SYNOPSIS**

42609 XSI
```
#include <signal.h>

int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```
42610

42611

42612

42613 **DESCRIPTION**

42614 Refer to *sighold*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sigsetjmp — set jump point for a non-local goto

SYNOPSIS

CX
```
#include <setjmp.h>

int sigsetjmp(sigjmp_buf env, int savemask);
```

DESCRIPTION

The *sigsetjmp*( ) function shall be equivalent to the *setjmp*( ) function, except as follows:

- References to *setjmp*( ) are equivalent to *sigsetjmp*( ).
- References to *longjmp*( ) are equivalent to *siglongjmp*( ).
- If the value of the *savemask* argument is not 0, *sigsetjmp*( ) shall also save the current signal mask of the calling thread as part of the calling environment.

RETURN VALUE

If the return is from a successful direct invocation, *sigsetjmp*( ) shall return 0. If the return is from a call to *siglongjmp*( ), *sigsetjmp*( ) shall return a non-zero value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The distinction between *setjmp*( )/*longjmp*( ) and *sigsetjmp*( )/*siglongjmp*( ) is only significant for programs which use *sigaction*( ), *sigprocmask*( ), or *sigsuspend*( ).

Note that since this function is defined in terms of *setjmp*( ), if *savemask* is zero, it is unspecified whether the signal mask is saved.

RATIONALE

The ISO C standard specifies various restrictions on the usage of the *setjmp*( ) macro in order to permit implementors to recognize the name in the compiler and not implement an actual function. These same restrictions apply to the *sigsetjmp*( ) macro.

There are processors that cannot easily support these calls, but this was not considered a sufficient reason to exclude them.

4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *_setjmp*( ) and *_longjmp*( ) that, together with *setjmp*( ) and *longjmp*( ), provide the same functionality as *sigsetjmp*( ) and *siglongjmp*( ). On those systems, *setjmp*( ) and *longjmp*( ) save and restore signal masks, while *_setjmp*( ) and *_longjmp*( ) do not. On System V Release 3 and in corresponding issues of the SVID, *setjmp*( ) and *longjmp*( ) are explicitly defined not to save and restore signal masks. In order to permit existing practice in both cases, the relation of *setjmp*( ) and *longjmp*( ) to signal masks is not specified, and a new set of functions is defined instead.

The *longjmp*( ) and *siglongjmp*( ) functions operate as in the previous issue provided the matching *setjmp*( ) or *sigsetjmp*( ) has been performed in the same thread. Non-local jumps into contexts saved by other threads would be at best a questionable practice and were not considered worthy of standardization.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

siglongjmp( ), *signal*( ), *sigprocmask*( ), *sigsuspend*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<setjmp.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The DESCRIPTION is reworded in terms of *setjmp*( ).

The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sigsuspend — wait for a signal

SYNOPSIS

CX
```
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);
```

DESCRIPTION

The *sigsuspend*( ) function shall replace the current signal mask of the calling thread with the set of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. This shall not cause any other signals that may have been pending on the process to become pending on the thread.

If the action is to terminate the process then *sigsuspend*( ) shall never return. If the action is to execute a signal-catching function, then *sigsuspend*( ) shall return after the signal-catching function returns, with the signal mask restored to the set that existed prior to the *sigsuspend*( ) call.

It is not possible to block signals that cannot be ignored. This is enforced by the system without causing an error to be indicated.

RETURN VALUE

Since *sigsuspend*( ) suspends thread execution indefinitely, there is no successful completion return value. If a return occurs, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sigsuspend*( ) function shall fail if:

[EINTR] A signal is caught by the calling process and control is returned from the signal-catching function.

EXAMPLES

None.

APPLICATION USAGE

Normally, at the beginning of a critical code section, a specified set of signals is blocked using the *sigprocmask*( ) function. When the thread has completed the critical section and needs to wait for the previously blocked signal(s), it pauses by calling *sigsuspend*( ) with the mask that was returned by the *sigprocmask*( ) call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 28), *pause*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The text in the RETURN VALUE section has been changed from ‘‘suspends process execution’’ to ‘‘suspends thread execution’’. This reflects IEEE PASC Interpretation 1003.1c #40.

Text in the APPLICATION USAGE section has been replaced.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sigtimedwait, sigwaitinfo — wait for queued signals (**REALTIME**)

SYNOPSIS

RTS
```
#include <signal.h>

int sigtimedwait(const sigset_t *restrict set,
    siginfo_t *restrict info,
    const struct timespec *restrict timeout);
int sigwaitinfo(const sigset_t *restrict set,
    siginfo_t *restrict info);
```

DESCRIPTION

The *sigtimedwait*( ) function shall be equivalent to *sigwaitinfo*( ) except that if none of the signals specified by *set* are pending, *sigtimedwait*( ) shall wait for the time interval specified in the **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-valued and if none of the signals specified by *set* are pending, then *sigtimedwait*( ) shall return immediately with an error. If *timeout* is the NULL pointer, the behavior is unspecified. If the Monotonic Clock option is supported, the CLOCK_MONOTONIC clock shall be used to measure the time interval specified by the *timeout* argument.

MON

The *sigwaitinfo*( ) function selects the pending signal from the set specified by *set*. Should any of multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at the time of the call, the calling thread shall be suspended until one or more signals in *set* become pending or until it is interrupted by an unblocked, caught signal.

The *sigwaitinfo*( ) function shall be equivalent to the *sigwait*( ) function if the *info* argument is NULL. If the *info* argument is non-NULL, the *sigwaitinfo*( ) function shall be equivalent to *sigwait*( ), except that the selected signal number shall be stored in the *si_signo* member, and the cause of the signal shall be stored in the *si_code* member. If any value is queued to the selected signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the value shall be stored in the *si_value* member of *info*. The system resource used to queue the signal shall be released and returned to the system for other use. If no value is queued, the content of the *si_value* member is undefined. If no further signals are queued for the selected signal, the pending indication for that signal shall be reset.

RETURN VALUE

Upon successful completion (that is, one of the signals specified by *set* is pending or is generated) *sigwaitinfo*( ) and *sigtimedwait*( ) shall return the selected signal number. Otherwise, the function shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sigtimedwait*( ) function shall fail if:

[EAGAIN] No signal specified by *set* was generated within the specified timeout period.

The *sigtimedwait*( ) and *sigwaitinfo*( ) functions may fail if:

[EINTR] The wait was interrupted by an unblocked, caught signal. It shall be documented in system documentation whether this error causes these functions to fail.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *sigtimedwait*( ) function may also fail if:

[EINVAL] The *timeout* argument specified a *tv_nsec* value less than zero or greater than or equal to 1 000 million.

An implementation only checks for this error if no signal is pending in *set* and it is necessary to wait.

EXAMPLES

None.

APPLICATION USAGE

The *sigtimedwait*( ) function times out and returns an [EAGAIN] error. Application writers should note that this is inconsistent with other functions such as *pthread_cond_timedwait*( ) that return [ETIMEDOUT].

RATIONALE

Existing programming practice on realtime systems uses the ability to pause waiting for a selected set of events and handle the first event that occurs in-line instead of in a signal-handling function. This allows applications to be written in an event-directed style similar to a state machine. This style of programming is useful for largescale transaction processing in which the overall throughput of an application and the ability to clearly track states are more important than the ability to minimize the response time of individual event handling.

It is possible to construct a signal-waiting macro function out of the realtime signal function mechanism defined in this volume of IEEE Std 1003.1-2001. However, such a macro has to include the definition of a generalized handler for all signals to be waited on. A significant portion of the overhead of handler processing can be avoided if the signal-waiting function is provided by the kernel. This volume of IEEE Std 1003.1-2001 therefore provides two signal-waiting functions—one that waits indefinitely and one with a timeout—as part of the overall realtime signal function specification.

The specification of a function with a timeout allows an application to be written that can be broken out of a wait after a set period of time if no event has occurred. It was argued that setting a timer event before the wait and recognizing the timer event in the wait would also implement the same functionality, but at a lower performance level. Because of the performance degradation associated with the user-level specification of a timer event and the subsequent cancellation of that timer event after the wait completes for a valid event, and the complexity associated with handling potential race conditions associated with the user-level method, the separate function has been included.

Note that the semantics of the *sigwaitinfo*( ) function are nearly identical to that of the *sigwait*( ) function defined by this volume of IEEE Std 1003.1-2001. The only difference is that *sigwaitinfo*( ) returns the queued signal value in the *value* argument. The return of the queued value is required so that applications can differentiate between multiple events queued to the same signal number.

The two distinct functions are being maintained because some implementations may choose to implement the POSIX Threads Extension functions and not implement the queued signals extensions. Note, though, that *sigwaitinfo*( ) does not return the queued value if the *value* argument is NULL, so the POSIX Threads Extension *sigwait*( ) function can be implemented as a macro on *sigwaitinfo*( ).

The *sigtimedwait*( ) function was separated from the *sigwaitinfo*( ) function to address concerns regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed wait, and immediate return, and concerns regarding consistency with other functions where the conditional and timed waits were separate functions from the pure blocking function. The

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

semantics of *sigtimedwait*( ) are specified such that *sigwaitinfo*( ) could be implemented as a macro with a NULL pointer for *timeout*.

The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-generated signals. One important question was how many threads that are suspended in a call to a *sigwait*( ) function for a signal should return from the call when the signal is sent. Four choices were considered:

1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
2. One or more threads return.
3. All waiting threads return.
4. Exactly one thread returns.

Prohibiting multiple calls to *sigwait*( ) for the same signal was felt to be overly restrictive. The ''one or more'' behavior made implementation of conforming packages easy at the expense of forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait*( ) in application code in order to achieve predictable behavior. There was concern that the ''all waiting threads'' behavior would result in ''signal broadcast storms'', consuming excessive CPU resources by replicating the signals in the general case. Furthermore, no convincing examples could be presented that delivery to all was either simpler or more powerful than delivery to one.

Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait* function for a signal should return when that signal occurs. This is not an onerous restriction as:

- A multi-way signal wait can be built from the single-way wait.
- Signals should only be handled by application-level code, as library routines cannot guess what the application wants to do with signals generated for the entire process.
- Applications can thus arrange for a single thread to wait for any given signal and call any needed routines upon its arrival.

In an application that is using signals for interprocess communication, signal processing is typically done in one place. Alternatively, if the signal is being caught so that process cleanup can be done, the signal handler thread can call separate process cleanup routines for each portion of the application. Since the application main line started each portion of the application, it is at the right abstraction level to tell each portion of the application to clean up.

Certainly, there exist programming styles where it is logical to consider waiting for a single signal in multiple threads. A simple *sigwait_multiple*( ) routine can be constructed to achieve this goal. A possible implementation would be to have each *sigwait_multiple*( ) caller registered as having expressed interest in a set of signals. The caller then waits on a thread-specific condition variable. A single server thread calls a *sigwait*( ) function on the union of all registered signals. When the *sigwait*( ) function returns, the appropriate state is set and condition variables are broadcast. New *sigwait_multiple*( ) callers may cause the pending *sigwait*( ) call to be canceled and reissued in order to update the set of signals being waited for.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.8.1 (on page 41), *pause*( ), *pthread_sigmask*( ), *sigaction*( ), *sigpending*( ), *sigsuspend*( ), *sigwait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<time.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

42851 **CHANGE HISTORY**

42852 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
42853 POSIX Threads Extension.

42854 **Issue 6**

42855 These functions are marked as part of the Realtime Signals Extension option.

42856 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo*() function
42857 has been corrected so that the second argument is of type **siginfo_t *****.

42858 The [ENOSYS] error condition has been removed as stubs need not be provided if an
42859 implementation does not support the Realtime Signals Extension option.

42860 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
42861 CLOCK_MONOTONIC clock, if supported, is used to measure timeout intervals.

42862 The **restrict** keyword is added to the *sigtimedwait*() and *sigwaitinfo*() prototypes for alignment
42863 with the ISO/IEC 9899:1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sigwait — wait for queued signals

SYNOPSIS

CX
```
#include <signal.h>

int sigwait(const sigset_t *restrict set, int *restrict sig);
```

DESCRIPTION

The *sigwait*( ) function shall select a pending signal from *set*, atomically clear it from the system's set of pending signals, and return that signal number in the location referenced by *sig*. If prior to the call to *sigwait*( ) there are multiple pending instances of a single signal number, it is implementation-defined whether upon successful return there are any remaining pending signals for that signal number. RTS If the implementation supports queued signals and there are multiple signals queued for the signal number selected, the first such queued signal shall cause a return from *sigwait*( ) and the remainder shall remain queued. If no signal in *set* is pending at the time of the call, the thread shall be suspended until one or more becomes pending. The signals defined by *set* shall have been blocked at the time of the call to *sigwait*( ); otherwise, the behavior is undefined. The effect of *sigwait*( ) on the signal actions for the signals in *set* is unspecified.

If more than one thread is using *sigwait*( ) to wait for the same signal, no more than one of these threads shall return from *sigwait*( ) with the signal number. Which thread returns from *sigwait*( ) if more than a single thread is waiting is unspecified.

RTS Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.

RETURN VALUE

Upon successful completion, *sigwait*( ) shall store the signal number of the received signal at the location referenced by *sig* and return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *sigwait*( ) function may fail if:

[EINVAL] The *set* argument contains an invalid or unsupported signal number.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

To provide a convenient way for a thread to wait for a signal, this volume of IEEE Std 1003.1-2001 provides the *sigwait*( ) function. For most cases where a thread has to wait for a signal, the *sigwait*( ) function should be quite convenient, efficient, and adequate.

However, requests were made for a lower-level primitive than *sigwait*( ) and for semaphores that could be used by threads. After some consideration, threads were allowed to use semaphores and *sem_post*( ) was defined to be async-signal and async-cancel-safe.

In summary, when it is necessary for code run in response to an asynchronous signal to notify a thread, *sigwait*( ) should be used to handle the signal. Alternatively, if the implementation provides semaphores, they also can be used, either following *sigwait*( ) or from within a signal handling routine previously registered with *sigaction*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 28), Section 2.8.1 (on page 41), *pause*( ), *pthread_sigmask*( ), *sigaction*( ), *sigpending*( ), *sigsuspend*( ), *sigwaitinfo*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The **restrict** keyword is added to the *sigwait*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sigwaitinfo — wait for queued signals (**REALTIME**)

SYNOPSIS

RTS

```
#include <signal.h>

int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);
```

DESCRIPTION

Refer to *sigtimedwait*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sin, sinf, sinl — sine function

SYNOPSIS

```
#include <math.h>

double sin(double x);
float sinf(float x);
long double sinl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the sine of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the sine of *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX

Domain Error	The *x* argument is ±Inf.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

MX

Range Error	The value of *x* is subnormal
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Taking the Sine of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45.0 * M_PI / 180;
double result;
...
result = sin(radians);
```

APPLICATION USAGE

These functions may lose accuracy when their argument is near a multiple of π or is far from 0.0.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asin( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes in previous issues.

Issue 6

The *sinf*( ) and *sinl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sinh, sinhf, sinhl — hyperbolic sine functions

SYNOPSIS

```
#include <math.h>

double sinh(double x);
float sinhf(float x);
long double sinhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the hyperbolic sine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic sine of *x*.

If the result would cause an overflow, a range error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for the type of the function.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions shall fail if:

Range Error — The result would cause an overflow.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

MX Range Error — The value *x* is subnormal.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asinh( ), *cosh*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tanh*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *sinhf*( ) and *sinhl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

43056 **NAME**

43057 sinl — sine function

43058 **SYNOPSIS**

43059 `#include <math.h>`

43060 `long double sinl(long double x);`

43061 **DESCRIPTION**

43062 Refer to *sin*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sleep — suspend execution for an interval of time

SYNOPSIS

```
#include <unistd.h>

unsigned sleep(unsigned seconds);
```

DESCRIPTION

The *sleep*( ) function shall cause the calling thread to be suspended from execution until either the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested due to the scheduling of other activity by the system.

If a SIGALRM signal is generated for the calling process during execution of *sleep*( ) and if the SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep*( ) returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also unspecified whether it remains pending after *sleep*( ) returns or it is discarded.

If a SIGALRM signal is generated for the calling process during execution of *sleep*( ), except as a result of a prior call to *alarm*( ), and if the SIGALRM signal is not being ignored or blocked from delivery, it is unspecified whether that signal has any effect other than causing *sleep*( ) to return.

If a signal-catching function interrupts *sleep*( ) and examines or changes either the time a SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or whether the SIGALRM signal is blocked from delivery, the results are unspecified.

If a signal-catching function interrupts *sleep*( ) and calls *siglongjmp*( ) or *longjmp*( ) to restore an environment saved prior to the *sleep*( ) call, the action associated with the SIGALRM signal and the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored as part of the environment.

XSI Interactions between *sleep*( ) and any of *setitimer*( ), *ualarm*( ), or *usleep*( ) are unspecified.

RETURN VALUE

If *sleep*( ) returns because the requested time has elapsed, the value returned shall be 0. If *sleep*( ) returns due to delivery of a signal, the return value shall be the ''unslept'' amount (the requested time minus the time actually slept) in seconds.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

There are two general approaches to the implementation of the *sleep*( ) function. One is to use the *alarm*( ) function to schedule a SIGALRM signal and then suspend the process waiting for that signal. The other is to implement an independent facility. This volume of IEEE Std 1003.1-2001 permits either approach.

In order to comply with the requirement that no primitive shall change a process attribute unless explicitly described by this volume of IEEE Std 1003.1-2001, an implementation using SIGALRM must carefully take into account any SIGALRM signal scheduled by previous *alarm*( ) calls, the

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

action previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has been scheduled before the *sleep*( ) would ordinarily complete, the *sleep*( ) must be shortened to that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-catching function) before *sleep*( ) returns. If a SIGALRM has been scheduled after the *sleep*( ) would ordinarily complete, it must be rescheduled for the same time before *sleep*( ) returns. The action and blocking for SIGALRM must be saved and restored.

Historical implementations often implement the SIGALRM-based version using *alarm*( ) and *pause*( ). One such implementation is prone to infinite hangups, as described in *pause*( ). Another such implementation uses the C-language *setjmp*( ) and *longjmp*( ) functions to avoid that window. That implementation introduces a different problem: when the SIGALRM signal interrupts a signal-catching function installed by the user to catch a different signal, the *longjmp*( ) aborts that signal-catching function. An implementation based on *sigprocmask*( ), *alarm*( ), and *sigsuspend*( ) can avoid these problems.

Despite all reasonable care, there are several very subtle, but detectable and unavoidable, differences between the two types of implementations. These are the cases mentioned in this volume of IEEE Std 1003.1-2001 where some other activity relating to SIGALRM takes place, and the results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of concern to most applications.

See also the discussion of the term *realtime* in *alarm*( ).

Since *sleep*( ) can be implemented using *alarm*( ), the discussion about alarms occurring early under *alarm*( ) applies to *sleep*( ) as well.

Application writers should note that the type of the argument *seconds* and the return value of *sleep*( ) is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the ISO C standard sets as 65 535, and any application passing a larger value is restricting its portability. A different type was considered, but historical implementations, including those with a 16-bit **int** type, consistently use either **unsigned** or **int**.

Scheduling delays may cause the process to return from the *sleep*( ) function significantly after the requested time. In such cases, the return value should be set to zero, since the formula (requested time minus the time actually spent) yields a negative number and *sleep*( ) returns an **unsigned**.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *getitimer*( ), *nanosleep*( ), *pause*( ), *sigaction*( ), *sigsetjmp*( ), *ualarm*( ), *usleep*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

43148 **NAME**

43149 snprintf — print formatted output

43150 **SYNOPSIS**

43151 `#include <stdio.h>`

43152 `int snprintf(char *restrict s, size_t n,`
43153 `    const char *restrict format, ...);`

43154 **DESCRIPTION**

43155 Refer to *fprintf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

sockatmark — determine whether a socket is at the out-of-band mark

SYNOPSIS

```
#include <sys/socket.h>

int sockatmark(int s);
```

DESCRIPTION

The *sockatmark*( ) function shall determine whether the socket specified by the descriptor *s* is at the out-of-band data mark (see the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.10.12, Socket Out-of-Band Data State). If the protocol for the socket supports out-of-band data by marking the stream with an out-of-band data mark, the *sockatmark*( ) function shall return 1 when all data preceding the mark has been read and the out-of-band data mark is the first element in the receive queue. The *sockatmark*( ) function shall not remove the mark from the stream.

RETURN VALUE

Upon successful completion, the *sockatmark*( ) function shall return a value indicating whether the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data preceding the mark has been read, the return value shall be 1; if there is no mark, or if data precedes the mark in the receive queue, the *sockatmark*( ) function shall return 0. Otherwise, it shall return a value of –1 and set *errno* to indicate the error.

ERRORS

The *sockatmark*( ) function shall fail if:

[EBADF] The *s* argument is not a valid file descriptor.

[ENOTTY] The *s* argument does not specify a descriptor for a socket.

EXAMPLES

None.

APPLICATION USAGE

The use of this function between receive operations allows an application to determine which received data precedes the out-of-band data and which follows the out-of-band data.

There is an inherent race condition in the use of this function. On an empty receive queue, the current read of the location might well be at the ‘‘mark’’, but the system has no way of knowing that the next data segment that will arrive from the network will carry the mark, and *sockatmark*( ) will return false, and the next read operation will silently consume the mark.

Hence, this function can only be used reliably when the application already knows that the out-of-band data has been seen by the system or that it is known that there is data waiting to be read at the socket (via SIGURG or *select*( )). See Section 2.10.11 (on page 61), Section 2.10.12 (on page 61), Section 2.10.14 (on page 62), and *pselect*( ) for details.

RATIONALE

The *sockatmark*( ) function replaces the historical SIOCATMARK command to *ioctl*( ) which implemented the same functionality on many implementations. Using a wrapper function follows the adopted conventions to avoid specifying commands to the *ioctl*( ) function, other than those now included to support XSI STREAMS. The *sockatmark*( ) function could be implemented as follows:

```
#include <sys/ioctl.h>

int sockatmark(int s)
{
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
43201           int val;
43202           if (ioctl(s,SIOCATMARK,&val)==-1)
43203               return(-1);
43204           return(val);
43205       }
```

43206 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of
43207 SIOCATMARK.

43208 **FUTURE DIRECTIONS**

43209 None.

43210 **SEE ALSO**

43211 *pselect*( ), *recv*( ), *recvmsg*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

43212 **CHANGE HISTORY**

43213 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

43214 **NAME**

43215 socket — create an endpoint for communication

43216 **SYNOPSIS**

43217 `#include <sys/socket.h>`

43218 `int socket(int domain, int type, int protocol);`

43219 **DESCRIPTION**

43220 The *socket*( ) function shall create an unbound socket in a communications domain, and return a
43221 file descriptor that can be used in later function calls that operate on sockets.

43222 The *socket*( ) function takes the following arguments:

43223 *domain* Specifies the communications domain in which a socket is to be created.

43224 *type* Specifies the type of socket to be created.

43225 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*
43226 of 0 causes *socket*( ) to use an unspecified default protocol appropriate for the
43227 requested socket type.

43228 The *domain* argument specifies the address family used in the communications domain. The
43229 address families supported by the system are implementation-defined.

43230 Symbolic constants that can be used for the domain argument are defined in the **<sys/socket.h>**
43231 header.

43232 The *type* argument specifies the socket type, which determines the semantics of communication
43233 over the socket. The following socket types are defined; implementations may specify additional
43234 socket types:

43235 SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte
43236 streams, and may provide a transmission mechanism for out-of-band
43237 data.

43238 SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages
43239 of fixed maximum length.

43240 SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode
43241 transmission paths for records. A record can be sent using one or more
43242 output operations and received using one or more input operations, but a
43243 single operation never transfers part of more than one record. Record
43244 boundaries are visible to the receiver via the MSG_EOR flag.

43245 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
43246 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
43247 be used. The protocols supported by the system are implementation-defined.

43248 The process may need to have appropriate privileges to use the *socket*( ) function or to create
43249 some sockets.

43250 **RETURN VALUE**

43251 Upon successful completion, *socket*( ) shall return a non-negative integer, the socket file
43252 descriptor. Otherwise, a value of –1 shall be returned and *errno* set to indicate the error.

43253 **ERRORS**

43254 The *socket*( ) function shall fail if:

43255 [EAFNOSUPPORT]
43256 The implementation does not support the specified address family.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

[EMFILE] No more file descriptors are available for this process.

[ENFILE] No more file descriptors are available for the system.

[EPROTONOSUPPORT]
The protocol is not supported by the address family, or the protocol is not supported by the implementation.

[EPROTOTYPE] The socket type is not supported by the protocol.

The *socket*( ) function may fail if:

[EACCES] The process does not have appropriate privileges.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENOMEM] Insufficient memory was available to fulfill the request.

EXAMPLES

None.

APPLICATION USAGE

The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.

The application can determine whether an address family is supported by trying to create a socket with *domain* set to the protocol in question.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *bind*( ), *connect*( ), *getsockname*( ), *getsockopt*( ), *listen*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *setsockopt*( ), *shutdown*( ), *socketpair*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<netinet/in.h>**, **<sys/socket.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

socketpair — create a pair of connected sockets

SYNOPSIS

```
#include <sys/socket.h>

int socketpair(int domain, int type, int protocol,
    int socket_vector[2]);
```

DESCRIPTION

The *socketpair*( ) function shall create an unbound pair of connected sockets in a specified *domain*, of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two sockets shall be identical. The file descriptors used in referencing the created sockets shall be returned in *socket_vector*[0] and *socket_vector*[1].

The *socketpair*( ) function takes the following arguments:

domain Specifies the communications domain in which the sockets are to be created.

type Specifies the type of sockets to be created.

protocol Specifies a particular protocol to be used with the sockets. Specifying a *protocol* of 0 causes *socketpair*( ) to use an unspecified default protocol appropriate for the requested socket type.

socket_vector Specifies a 2-integer array to hold the file descriptors of the created socket pair.

The *type* argument specifies the socket type, which determines the semantics of communications over the socket. The following socket types are defined; implementations may specify additional socket types:

SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.

SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.

SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode transmission paths for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.

If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address family. If the *protocol* argument is zero, the default protocol for this address family and type shall be used. The protocols supported by the system are implementation-defined.

The process may need to have appropriate privileges to use the *socketpair*( ) function or to create some sockets.

RETURN VALUE

Upon successful completion, this function shall return 0; otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *socketpair*( ) function shall fail if:

[EAFNOSUPPORT]
The implementation does not support the specified address family.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

43329 [EMFILE] No more file descriptors are available for this process.

43330 [ENFILE] No more file descriptors are available for the system.

43331 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.

43332 [EPROTONOSUPPORT]
43333 The protocol is not supported by the address family, or the protocol is not
43334 supported by the implementation.

43335 [EPROTOTYPE] The socket type is not supported by the protocol.

43336 The *socketpair*( ) function may fail if:

43337 [EACCES] The process does not have appropriate privileges.

43338 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

43339 [ENOMEM] Insufficient memory was available to fulfill the request.

43340 **EXAMPLES**

43341 None.

43342 **APPLICATION USAGE**

43343 The documentation for specific address families specifies which protocols each address family
43344 supports. The documentation for specific protocols specifies which socket types each protocol
43345 supports.

43346 The *socketpair*( ) function is used primarily with UNIX domain sockets and need not be
43347 supported for other domains.

43348 **RATIONALE**

43349 None.

43350 **FUTURE DIRECTIONS**

43351 None.

43352 **SEE ALSO**

43353 *socket*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/socket.h>**

43354 **CHANGE HISTORY**

43355 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

43356 **NAME**

43357 sprintf — print formatted output

43358 **SYNOPSIS**

43359 `#include <stdio.h>`

43360 `int sprintf(char *restrict s, const char *restrict format, ...);`

43361 **DESCRIPTION**

43362 Refer to *fprintf*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

sqrt, sqrtf, sqrtl — square root function

SYNOPSIS

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the square root of their argument *x*, $\sqrt{x}$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the square root of *x*.

MX For finite values of $x < -0$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or +Inf, *x* shall be returned.

If *x* is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX Domain Error — The finite value of *x* is < –0, or *x* is –Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

EXAMPLES

Taking the Square Root of 9.0

```
#include <math.h>
...
double x = 9.0;
double result;
...
result = sqrt(x);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

43402 **APPLICATION USAGE**

43403 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
43404 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

43405 **RATIONALE**

43406 None.

43407 **FUTURE DIRECTIONS**

43408 None.

43409 **SEE ALSO**

43410 *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
43411 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**, **<stdio.h>**

43412 **CHANGE HISTORY**

43413 First released in Issue 1. Derived from Issue 1 of the SVID.

43414 **Issue 5**

43415 The DESCRIPTION is updated to indicate how an application should check for an error. This
43416 text was previously published in the APPLICATION USAGE section.

43417 **Issue 6**

43418 The *sqrtf*( ) and *sqrtl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

43419 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
43420 revised to align with the ISO / IEC 9899: 1999 standard.

43421 IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are
43422 marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

43423 **NAME**

43424 srand — pseudo-random number generator

43425 **SYNOPSIS**

43426 `#include <stdlib.h>`

43427 `void srand(unsigned` *`seed`*`);`

43428 **DESCRIPTION**

43429 Refer to *rand*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

srand48 — seed the uniformly distributed double-precision pseudo-random number generator

SYNOPSIS

XSI

```
#include <stdlib.h>

void srand48(long seedval);
```

DESCRIPTION

Refer to *drand48*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

srandom — seed pseudo-random number generator

SYNOPSIS

XSI

```
#include <stdlib.h>

void srandom(unsigned seed);
```

DESCRIPTION

Refer to *initstate*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

43446 **NAME**

43447 sscanf — convert formatted input

43448 **SYNOPSIS**

43449 `#include <stdio.h>`

43450 `int sscanf(const char *restrict s, const char *restrict format, ...);`

43451 **DESCRIPTION**

43452 Refer to *fscanf*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

stat — get file status

SYNOPSIS

```
#include <sys/stat.h>

int stat(const char *restrict path, struct stat *restrict buf);
```

DESCRIPTION

The *stat*( ) function shall obtain information about the named file and write it to the area pointed to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. An implementation that provides additional or alternate file access control mechanisms may, under implementation-defined conditions, cause *stat*( ) to fail. In particular, the system may deny the existence of the file specified by *path*.

If the named file is a symbolic link, the *stat*( ) function shall continue pathname resolution using the contents of the symbolic link, and shall return information pertaining to the resulting file if the file exists.

The *buf* argument is a pointer to a **stat** structure, as defined in the **<sys/stat.h>** header, into which information is placed concerning the file.

The *stat*( ) function shall update any time-related fields (as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.7, File Times Update), before writing into the **stat** structure.

Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*, *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in this volume of IEEE Std 1003.1-2001. The value of the member *st_nlink* shall be set to the number of links to the file.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *stat*( ) function shall fail if:

[EACCES] Search permission is denied for a component of the path prefix.

[EIO] An error occurred while reading from the file system.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

[EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *stat*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

[EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.

EXAMPLES

Obtaining File Status Information

The following example shows how to obtain file status information for a file named **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct stat buffer;
int         status;
...
status = stat("/home/cnd/mod1", &buffer);
```

Getting Directory Information

The following example fragment gets status information for each entry in a directory. The call to the *stat*( ) function stores file information in the **stat** structure pointed to by *statbuf*. The lines that follow the *stat*( ) call format the fields in the **stat** structure for presentation to the user of the program.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <locale.h>
#include <langinfo.h>
#include <stdio.h>
#include <stdint.h>

struct dirent  *dp;
struct stat     statbuf;
struct passwd  *pwd;
struct group   *grp;
struct tm      *tm;
char            datestring[256];
...
/* Loop through directory entries. */
while ((dp = readdir(dir)) != NULL) {

    /* Get entry's information. */
    if (stat(dp->d_name, &statbuf) == -1)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
43538                continue;
43539            /* Print out type, permissions, and number of links. */
43540            printf("%10.10s", sperm (statbuf.st_mode));
43541            printf("%4d", statbuf.st_nlink);
43542            /* Print out owner's name if it is found using getpwuid(). */
43543            if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
43544                printf(" %-8.8s", pwd->pw_name);
43545            else
43546                printf(" %-8d", statbuf.st_uid);
43547            /* Print out group name if it is found using getgrgid(). */
43548            if ((grp = getgrgid(statbuf.st_gid)) != NULL)
43549                printf(" %-8.8s", grp->gr_name);
43550            else
43551                printf(" %-8d", statbuf.st_gid);
43552            /* Print size of file. */
43553            printf(" %9jd", (intmax_t)statbuf.st_size);
43554            tm = localtime(&statbuf.st_mtime);
43555            /* Get localized date string. */
43556            strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
43557            printf(" %s %s\n", datestring, dp->d_name);
43558        }
```

43559 **APPLICATION USAGE**

43560 None.

43561 **RATIONALE**

43562 The intent of the paragraph describing ''additional or alternate file access control mechanisms''
43563 is to allow a secure implementation where a process with a label that does not dominate the
43564 file's label cannot perform a *stat*( ) function. This is not related to read permission; a process with
43565 a label that dominates the file's label does not need read permission. An implementation that
43566 supports write-up operations could fail *fstat*( ) function calls even though it has a valid file
43567 descriptor open for writing.

43568 **FUTURE DIRECTIONS**

43569 None.

43570 **SEE ALSO**

43571 *fstat*( ), *lstat*( ), *readlink*( ), *symlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
43572 **<sys/stat.h>**, **<sys/types.h>**

43573 **CHANGE HISTORY**

43574 First released in Issue 1. Derived from Issue 1 of the SVID.

43575 **Issue 5**

43576 Large File Summit extensions are added.

43577 **Issue 6**

43578 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

43579 The following new requirements on POSIX implementations derive from alignment with the
43580 Single UNIX Specification:

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *stat*( ) prototype for alignment with the ISO/IEC 9899:1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

43596 **NAME**

43597 statvfs — get file system information

43598 **SYNOPSIS**

43599 XSI
```
#include <sys/statvfs.h>
```

43600
```
int statvfs(const char *restrict path, struct statvfs *restrict buf);
```

43601

43602 **DESCRIPTION**

43603 Refer to *fstatvfs*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

stderr, stdin, stdout — standard I/O streams

SYNOPSIS

```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type **FILE**. The *fopen*( ) function shall create certain descriptive data for a stream and return a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the **<stdio.h>** header and associated with the standard open files.

At program start-up, three streams shall be predefined and need not be opened explicitly: *standard input* (for reading conventional input), *standard output* (for writing conventional output), and *standard error* (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

CX The following symbolic values in **<unistd.h>** define the file descriptors that shall be associated with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

STDIN_FILENO	Standard input value, *stdin*. Its value is 0.
STDOUT_FILENO	Standard output value, *stdout*. Its value is 1.
STDERR_FILENO	Standard error value, *stderr*. Its value is 2.

The *stderr* stream is expected to be open for reading and writing.

RETURN VALUE

None.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose( ), *feof*( ), *ferror*( ), *fileno*( ), *fopen*( ), *fread*( ), *fseek*( ), *getc*( ), *gets*( ), *popen*( ), *printf*( ), *putc*( ), *puts*( ), *read*( ), *scanf*( ), *setbuf*( ), *setvbuf*( ), *tmpfile*( ), *ungetc*( ), *vprintf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<unistd.h>**

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

43644 **CHANGE HISTORY**

43645 First released in Issue 1.

43646 **Issue 6**

43647 Extensions beyond the ISO C standard are marked.

43648 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

strcasecmp, strncasecmp — case-insensitive string comparisons

SYNOPSIS

XSI
```
#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, size_t n);
```

DESCRIPTION

The *strcasecmp*( ) function shall compare, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasecmp*( ) function shall compare, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

In the POSIX locale, *strcasecmp*( ) and *strncasecmp*( ) shall behave as if the strings had been converted to lowercase and then a byte comparison performed. The results are unspecified in other locales.

RETURN VALUE

Upon completion, *strcasecmp*( ) shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to by *s2*, respectively.

Upon successful completion, *strncasecmp*( ) shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strcat — concatenate two strings

SYNOPSIS

```
#include <string.h>

char *strcat(char *restrict s1, const char *restrict s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strcat*( ) function shall append a copy of the string pointed to by *s2* (including the terminating null byte) to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites the null byte at the end of *s1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *strcat*( ) function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncat( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *strcat*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

strchr — string scanning operation

SYNOPSIS

```
#include <string.h>

char *strchr(const char *s, int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strchr*( ) function shall locate the first occurrence of *c* (converted to a **char**) in the string pointed to by *s*. The terminating null byte is considered to be part of the string.

RETURN VALUE

Upon completion, *strchr*( ) shall return a pointer to the byte, or a null pointer if the byte was not found.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strrchr( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strcmp — compare two strings

SYNOPSIS

```
#include <string.h>

int strcmp(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strcmp*( ) function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

RETURN VALUE

Upon completion, *strcmp*( ) shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

Checking a Password Entry

The following example compares the information read from standard input to the value of the name of the user entry. If the *strcmp*( ) function returns 0 (indicating a match), a further check will be made to see if the user entered the proper old password. The *crypt*( ) function shall encrypt the old password entered by the user, using the value of the encrypted password in the **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the structure, the entered password *oldpasswd* is the correct user's password. Finally, the program encrypts the new password so that it can store the information in the **passwd** structure.

```
#include <string.h>
#include <unistd.h>
#include <stdio.h>
...
int valid_change;
struct passwd *p;
char user[100];
char oldpasswd[100];
char newpasswd[100];
char savepasswd[100];
...
if (strcmp(p->pw_name, user) == 0) {
    if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
        strcpy(savepasswd, crypt(newpasswd, user));
        p->pw_passwd = savepasswd;
        valid_change = 1;
    }
    else {
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                fprintf(stderr, "Old password is not valid\n");
            }
        }
        ...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncmp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strcoll — string comparison using collating information

SYNOPSIS

```
#include <string.h>

int strcoll(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strcoll*( ) function shall compare the string pointed to by *s1* to the string pointed to by *s2*, both interpreted as appropriate to the *LC_COLLATE* category of the current locale.

CX The *strcoll*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strcoll*( ), then check *errno*.

RETURN VALUE

Upon successful completion, *strcoll*( ) shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* when both are interpreted as appropriate to the current locale. On error, *strcoll*( ) may set *errno*, but no return value is reserved to indicate an error.

ERRORS

The *strcoll*( ) function may fail if:

CX [EINVAL] The *s1* or *s2* arguments contain characters outside the domain of the collating sequence.

EXAMPLES

Comparing Nodes

The following example uses an application-defined function, *node_compare*( ), to compare two nodes based on an alphabetical ordering of the *string* field.

```
#include <string.h>
...
struct node { /* These are stored in the table. */
    char *string;
    int length;
};
...
int node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
        ((const struct node *)node2)->string);
}
...
```

APPLICATION USAGE

The *strxfrm*( ) and *strcmp*( ) functions should be used for sorting large lists.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

43853 **RATIONALE**

43854 None.

43855 **FUTURE DIRECTIONS**

43856 None.

43857 **SEE ALSO**

43858 *strcmp*( ), *strxfrm*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

43859 **CHANGE HISTORY**

43860 First released in Issue 3.

43861 **Issue 5**

43862 The DESCRIPTION is updated to indicate that *errno* does not change if the function is
43863 successful.

43864 **Issue 6**

43865 Extensions beyond the ISO C standard are marked.

43866 The following new requirements on POSIX implementations derive from alignment with the
43867 Single UNIX Specification:

43868 • The [EINVAL] optional error condition is added.

43869 An example is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strcpy — copy a string

SYNOPSIS

```
#include <string.h>

char *strcpy(char *restrict s1, const char *restrict s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strcpy*( ) function shall copy the string pointed to by *s2* (including the terminating null byte) into the array pointed to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *strcpy*( ) function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

Initializing a String

The following example copies the string `"----------"` into the *permstring* variable.

```
#include <string.h>
...
static char permstring[11];
...
strcpy(permstring, "----------");
...
```

Storing a Key and Data

The following example allocates space for a key using *malloc*( ) then uses *strcpy*( ) to place the key there. Then it allocates space for data using *malloc*( ), and uses *strcpy*( ) to place data there. (The user-defined function *dbfree*( ) frees memory previously allocated to an array of type **struct element ***.)

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
...
/* Structure used to read data and store it. */
struct element {
    char *key;
    char *data;
};

struct element *tbl, *curtbl;
char *key, *data;
int count;
...
void dbfree(struct element *, int);
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        ...
        if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
            perror("malloc"); dbfree(tbl, count); return NULL;
        }
        strcpy(curtbl->key, key);
        if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
            perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
        }
        strcpy(curtbl->data, data);
        ...
```

APPLICATION USAGE

Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncpy( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *strcpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

43939 **NAME**

43940 strcspn — get the length of a complementary substring

43941 **SYNOPSIS**

43942
```
#include <string.h>
```

43943
```
size_t strcspn(const char *s1, const char *s2);
```

43944 **DESCRIPTION**

43945 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43946 conflict between the requirements described here and the ISO C standard is unintentional. This
43947 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43948 The *strcspn*( ) function shall compute the length (in bytes) of the maximum initial segment of the
43949 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

43950 **RETURN VALUE**

43951 The *strcspn*( ) function shall return the length of the computed segment of the string pointed to
43952 by *s1*; no return value is reserved to indicate an error.

43953 **ERRORS**

43954 No errors are defined.

43955 **EXAMPLES**

43956 None.

43957 **APPLICATION USAGE**

43958 None.

43959 **RATIONALE**

43960 None.

43961 **FUTURE DIRECTIONS**

43962 None.

43963 **SEE ALSO**

43964 *strspn*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

43965 **CHANGE HISTORY**

43966 First released in Issue 1. Derived from Issue 1 of the SVID.

43967 **Issue 5**

43968 The RETURN VALUE section is updated to indicate that *strcspn*( ) returns the length of *s1*, and
43969 not *s1* itself as was previously stated.

43970 **Issue 6**

43971 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is
43972 updated to indicate that the computed segment length is returned, not the *s1* length.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strdup — duplicate a string

SYNOPSIS

XSI
```
#include <string.h>

char *strdup(const char *s1);
```

DESCRIPTION

The *strdup*( ) function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s1*. The returned pointer can be passed to *free*( ). A null pointer is returned if the new string cannot be created.

RETURN VALUE

The *strdup*( ) function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *strdup*( ) function may fail if:

[ENOMEM] Storage space available is insufficient.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *malloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strerror, strerror_r — get error message string

SYNOPSIS

```
#include <string.h>

char *strerror(int errnum);
int strerror_r(int errnum, char *strerrbuf, size_t buflen);
```

DESCRIPTION

CX For *strerror*( ): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strerror*( ) function shall map the error number in *errnum* to a locale-dependent error message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*, but *strerror*( ) shall map any value of type **int** to a message.

The string pointed to shall not be modified by the application, but may be overwritten by a CX subsequent call to *strerror*( ) or *perror*( ).

CX The contents of the error message strings returned by *strerror*( ) should be determined by the setting of the *LC_MESSAGES* category in the current locale.

The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *strerror*( ).

CX The *strerror*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strerror*( ), then check *errno*.

The *strerror*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

TSF The *strerror_r*( ) function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

RETURN VALUE

Upon successful completion, *strerror*( ) shall return a pointer to the generated message string. On error *errno* may be set, but no return value is reserved to indicate an error.

TSF Upon successful completion, *strerror_r*( ) shall return 0. Otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

[EINVAL] The value of *errnum* is not a valid error number.

The *strerror_r*( ) function may fail if:

TSF [ERANGE] Insufficient storage was supplied via *strerrbuf* and *buflen* to contain the generated message string.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

perror( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the fact that *errno* may be set is added.
- The [EINVAL] optional error condition is added.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The *strerror_r*( ) function is added in response to IEEE PASC Interpretation 1003.1c #39.

The *strerror_r*( ) function is marked as part of the Thread-Safe Functions option.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strfmon — convert monetary value to a string

SYNOPSIS

XSI
```
#include <monetary.h>

ssize_t strfmon(char *restrict s, size_t maxsize,
    const char *restrict format, ...);
```

DESCRIPTION

The *strfmon*( ) function shall place characters into the array pointed to by *s* as controlled by the string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

The format is a character string, beginning and ending in its initial state, if any, that contains two types of objects: *plain characters*, which are simply copied to the output stream, and *conversion specifications*, each of which shall result in the fetching of zero or more arguments which are converted and formatted. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored.

The application shall ensure that a conversion specification consists of the following sequence:

- A '%' character
- Optional flags
- Optional field width
- Optional left precision
- Optional right precision
- A required conversion specifier character that determines the conversion to be performed

Flags

One or more of the following optional flags can be specified to control the conversion:

=f An '=' followed by a single character *f* which is used as the numeric fill character. In order to work with precision or width counts, the fill character shall be a single byte character; if not, the behavior is undefined. The default numeric fill character is the <space>. This flag does not affect field width filling which always uses the <space>. This flag is ignored unless a left precision (see below) is specified.

^ Do not format the currency amount with grouping characters. The default is to insert the grouping characters if defined for the current locale.

+ or (Specify the style of representing positive and negative currency amounts. Only one of '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '−' are used (for example, in the U.S., the empty string if positive and '−' if negative). If '(' is specified, negative amounts are enclosed within parentheses. If neither flag is specified, the '+' style is used.

! Suppress the currency symbol from the output conversion.

− Specify the alignment. If this flag is present the result of the conversion is left-justified (padded to the right) rather than right-justified. This flag shall be ignored unless a field width (see below) is specified.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

44107 **Field Width**

44108 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result
44109 of the conversion is right-justified (or left-justified if the flag '−' is specified). The
44110 default is 0.

44111 **Left Precision**

44112 #*n* A '#' followed by a decimal digit string *n* specifying a maximum number of digits
44113 expected to be formatted to the left of the radix character. This option can be used to
44114 keep the formatted output from multiple calls to the *strfmon*( ) function aligned in the
44115 same columns. It can also be used to fill unused positions with a special character as in
44116 "$***123.45". This option causes an amount to be formatted as if it has the number
44117 of digits specified by *n*. If more than *n* digit positions are required, this conversion
44118 specification is ignored. Digit positions in excess of those actually required are filled
44119 with the numeric fill character (see the =*f* flag above).

44120 If grouping has not been suppressed with the '^' flag, and it is defined for the current
44121 locale, grouping separators are inserted before the fill characters (if any) are added.
44122 Grouping separators are not applied to fill characters even if the fill character is a digit.

44123 To ensure alignment, any characters appearing before or after the number in the
44124 formatted output such as currency or sign symbols are padded as necessary with
44125 <space>s to make their positive and negative formats an equal length.

44126 **Right Precision**

44127 .*p* A period followed by a decimal digit string *p* specifying the number of digits after the
44128 radix character. If the value of the right precision *p* is 0, no radix character appears. If a
44129 right precision is not included, a default specified by the current locale is used. The
44130 amount being formatted is rounded to the specified number of digits prior to
44131 formatting.

44132 **Conversion Specifier Characters**

44133 The conversion specifier characters and their meanings are:

44134 i The **double** argument is formatted according to the locale's international currency
44135 format (for example, in the U.S.: USD 1,234.56). If the argument is ±Inf or NaN, the
44136 result of the conversion is unspecified.

44137 n The **double** argument is formatted according to the locale's national currency format
44138 (for example, in the U.S.: $1,234.56). If the argument is ±Inf or NaN, the result of the
44139 conversion is unspecified.

44140 % Convert to a '%'; no argument is converted. The entire conversion specification shall
44141 be %%.

44142 **Locale Information**

44143 The *LC_MONETARY* category of the program's locale affects the behavior of this function
44144 including the monetary radix character (which may be different from the numeric radix
44145 character affected by the *LC_NUMERIC* category), the grouping separator, the currency
44146 symbols, and formats. The international currency symbol should be conformant with the
44147 ISO 4217: 2001 standard.

44148 If the value of *maxsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

RETURN VALUE

If the total number of resulting bytes including the terminating null byte is not more than *maxsize*, *strfmon*( ) shall return the number of bytes placed into the array pointed to by *s*, not including the terminating null byte. Otherwise, –1 shall be returned, the contents of the array are unspecified, and *errno* shall be set to indicate the error.

ERRORS

The *strfmon*( ) function shall fail if:

[E2BIG] Conversion stopped due to lack of space in the buffer.

EXAMPLES

Given a locale for the U.S. and the values 123.45, –123.45, and 3456.781, the following output might be produced. Square brackets ("`[]`") are used in this example to delimit the output.

`%n`	`[$123.45]`
	`[-$123.45]`
	`[$3,456.78]`
		Default formatting

`%11n`	`[    $123.45]`
	`[   -$123.45]`
	`[  $3,456.78]`
		Right align within an 11-character field

`%#5n`	`[ $   123.45]`
	`[-$   123.45]`
	`[ $ 3,456.78]`
		Aligned columns for values up to 99 999

`%=*#5n`	`[ $***123.45]`
	`[-$***123.45]`
	`[ $*3,456.78]`
		Specify a fill character

`%=0#5n`	`[ $000123.45]`
	`[-$000123.45]`
	`[ $03,456.78]`
		Fill characters do not use grouping even if the fill character is a digit

`%^#5n`	`[ $  123.45]`
	`[-$  123.45]`
	`[ $ 3456.78]`
		Disable the grouping separator

`%^#5.0n`	`[ $  123]`
	`[-$  123]`
	`[ $ 3457]`
		Round off to whole units

`%^#5.4n`	`[ $  123.4500]`
	`[-$  123.4500]`
	`[ $ 3456.7810]`
		Increase the precision

`%(#5n`	`[$   123.45]`
	`[($   123.45)]`
	`[$ 3,456.78]`
		Use an alternative pos/neg style

`%!(#5n`	`[   123.45]`
	`[(   123.45)]`
	`[ 3,456.78]`
		Disable the currency symbol

`%-14#5.4n`	`[ $   123.4500 ]`
	`[-$   123.4500 ]`
	`[ $ 3,456.7810 ]`
		Left-justify the output

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
%14#5.4n    [  $   123.4500]   Corresponding right-justified output
            [ -$   123.4500]
            [  $ 3,456.7810]
```

See also the EXAMPLES section in *fprintf*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

Lowercase conversion characters are reserved for future standards use and uppercase for implementation-defined use.

SEE ALSO

fprintf( ), *localeconv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<monetary.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Moved from ENHANCED I18N to BASE.

The [ENOSYS] error is removed.

A sentence is added to the DESCRIPTION warning about values of *maxsize* that are greater than {SSIZE_MAX}.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *strfmon*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

The EXAMPLES section is reworked, clarifying the output format.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strftime — convert date and time to a string

SYNOPSIS

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strftime*( ) function shall place bytes into the array pointed to by *s* as controlled by the string pointed to by *format*. The format is a character string, beginning and ending in its initial shift state, if any. The *format* string consists of zero or more conversion specifications and ordinary characters. A conversion specification consists of a `'%'` character, possibly followed by an `E` or `O` modifier, and a terminating conversion specifier character that determines the conversion specification's behavior. All ordinary characters (including the terminating null byte) are copied unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than *maxsize* bytes are placed into the array. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined using the *LC_TIME* category of the current locale and by the values of zero or more members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the description. If any of the specified values are outside the normal range, the characters stored are unspecified.

CX Local timezone information is used as though *strftime*( ) called *tzset*( ).

The following conversion specifications are supported:

`%a` Replaced by the locale's abbreviated weekday name. [*tm_wday*]

`%A` Replaced by the locale's full weekday name. [*tm_wday*]

`%b` Replaced by the locale's abbreviated month name. [*tm_mon*]

`%B` Replaced by the locale's full month name. [*tm_mon*]

`%c` Replaced by the locale's appropriate date and time representation. (See the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**.)

`%C` Replaced by the year divided by 100 and truncated to an integer, as a decimal number [00,99]. [*tm_year*]

`%d` Replaced by the day of the month as a decimal number [01,31]. [*tm_mday*]

`%D` Equivalent to `%m/%d/%y`. [*tm_mon*, *tm_mday*, *tm_year*]

`%e` Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space. [*tm_mday*]

`%F` Equivalent to `%Y-%m-%d` (the ISO 8601:2000 standard date format). [*tm_year*, *tm_mon*, *tm_mday*]

`%g` Replaced by the last 2 digits of the week-based year (see below) as a decimal number [00,99]. [*tm_year*, *tm_wday*, *tm_yday*]

`%G` Replaced by the week-based year (see below) as a decimal number (for example, 1977). [*tm_year*, *tm_wday*, *tm_yday*]

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

44261		%h	Equivalent to %b. [*tm_mon*]
44262		%H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [*tm_hour*]
44263		%I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [*tm_hour*]
44264		%j	Replaced by the day of the year as a decimal number [001,366]. [*tm_yday*]
44265		%m	Replaced by the month as a decimal number [01,12]. [*tm_mon*]
44266		%M	Replaced by the minute as a decimal number [00,59]. [*tm_min*]
44267		%n	Replaced by a <newline>.
44268		%p	Replaced by the locale's equivalent of either a.m. or p.m. [*tm_hour*]
44269	CX	%r	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be
44270			equivalent to %I:%M:%S %p. [*tm_hour*, *tm_min*, *tm_sec*]
44271		%R	Replaced by the time in 24-hour notation (%H:%M). [*tm_hour*, *tm_min*]
44272		%S	Replaced by the second as a decimal number [00,60]. [*tm_sec*]
44273		%t	Replaced by a <tab>.
44274		%T	Replaced by the time (%H:%M:%S). [*tm_hour*, *tm_min*, *tm_sec*]
44275		%u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday.
44276			[*tm_wday*]
44277		%U	Replaced by the week number of the year as a decimal number [00,53]. The first
44278			Sunday of January is the first day of week 1; days in the new year before this are in
44279			week 0. [*tm_year*, *tm_wday*, *tm_yday*]
44280		%V	Replaced by the week number of the year (Monday as the first day of the week) as a
44281			decimal number [01,53]. If the week containing 1 January has four or more days in the
44282			new year, then it is considered week 1. Otherwise, it is the last week of the previous
44283			year, and the next week is week 1. Both January 4th and the first Thursday of January
44284			are always in week 1. [*tm_year*, *tm_wday*, *tm_yday*]
44285		%w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.
44286			[*tm_wday*]
44287		%W	Replaced by the week number of the year as a decimal number [00,53]. The first
44288			Monday of January is the first day of week 1; days in the new year before this are in
44289			week 0. [*tm_year*, *tm_wday*, *tm_yday*]
44290		%x	Replaced by the locale's appropriate date representation. (See the Base Definitions
44291			volume of IEEE Std 1003.1-2001, **<time.h>**.)
44292		%X	Replaced by the locale's appropriate time representation. (See the Base Definitions
44293			volume of IEEE Std 1003.1-2001, **<time.h>**.)
44294		%y	Replaced by the last two digits of the year as a decimal number [00,99]. [*tm_year*]
44295		%Y	Replaced by the year as a decimal number (for example, 1997). [*tm_year*]
44296		%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or
44297			−hhmm), or by no characters if no timezone is determinable. For example, "−0430"
44298	CX		means 4 hours 30 minutes behind UTC (west of Greenwich). If *tm_isdst* is zero, the
44299			standard time offset is used. If *tm_isdst* is greater than zero, the daylight savings time
44300			offset is used. If *tm_isdst* is negative, no characters are returned. [*tm_isdst*]

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`%Z` Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [*tm_isdst*]

`%%` Replaced by `%`.

If a conversion specification does not correspond to any of the above, the behavior is undefined.

CX If a **struct tm** broken-down time structure is created by *localtime*( ) or *localtime_r*( ), or modified by *mktime*( ), and the value of *TZ* is subsequently modified, the results of the `%Z` and `%z` *strftime*( ) conversion specifiers are undefined, when *strftime*( ) is called with such a broken-down time structure.

If a **struct tm** broken-down time structure is created or modified by *gmtime*( ) or *gmtime_r*( ), it is unspecified whether the result of the `%Z` and `%z` conversion specifiers shall refer to UTC or the current local timezone, when *strftime*( ) is called with such a broken-down time structure.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the `E` or `O` modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5, LC_TIME), the behavior shall be as if the unmodified conversion specification were used.

`%Ec` Replaced by the locale's alternative appropriate date and time representation.

`%EC` Replaced by the name of the base year (period) in the locale's alternative representation.

`%Ex` Replaced by the locale's alternative date representation.

`%EX` Replaced by the locale's alternative time representation.

`%Ey` Replaced by the offset from `%EC` (year only) in the locale's alternative representation.

`%EY` Replaced by the full alternative year representation.

`%Od` Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading spaces.

`%Oe` Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.

`%OH` Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.

`%OI` Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.

`%Om` Replaced by the month using the locale's alternative numeric symbols.

`%OM` Replaced by the minutes using the locale's alternative numeric symbols.

`%OS` Replaced by the seconds using the locale's alternative numeric symbols.

`%Ou` Replaced by the weekday as a number in the locale's alternative representation (Monday=1).

`%OU` Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to `%U`) using the locale's alternative numeric symbols.

`%OV` Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to `%V`) using the locale's alternative numeric symbols.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`%Ow` Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.

`%OW` Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

`%Oy` Replaced by the year (offset from `%C`) using the locale's alternative numeric symbols.

`%g`, `%G`, and `%V` give values according to the ISO 8601:2000 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, `%G` is replaced by 1998 and `%V` is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, `%G` is replaced by 1998 and `%V` is replaced by 01.

If a conversion specifier is not one of the above, the behavior is undefined.

RETURN VALUE

If the total number of resulting bytes including the terminating null byte is not more than *maxsize*, *strftime*( ) shall return the number of bytes placed into the array pointed to by *s*, not including the terminating null byte. Otherwise, 0 shall be returned and the contents of the array are unspecified.

ERRORS

No errors are defined.

EXAMPLES

Getting a Localized Date String

The following example first sets the locale to the user's default. The locale information will be used in the *nl_langinfo*( ) and *strftime*( ) functions. The *nl_langinfo*( ) function returns the localized date string which specifies how the date is laid out. The *strftime*( ) function takes this information and, using the **tm** structure for values, places the date and time information into *datestring*.

```
#include <time.h>
#include <locale.h>
#include <langinfo.h>
...
struct tm *tm;
char datestring[256];
...
setlocale (LC_ALL, "");
...
strftime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
...
```

APPLICATION USAGE

The range of values for `%S` is [00,60] rather than [00,59] to allow for the occasional leap second.

Some of the conversion specifications are duplicates of others. They are included for compatibility with *nl_cxtime*( ) and *nl_ascxtime*( ), which were published in Issue 2.

Applications should use `%Y` (4-digit years) in preference to `%y` (2-digit years).

In the C locale, the `E` and `O` modifiers are ignored and the replacement strings for the following specifiers are:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

44386	`%a`	The first three characters of `%A`.
44387	`%A`	One of Sunday, Monday, ..., Saturday.
44388	`%b`	The first three characters of `%B`.
44389	`%B`	One of January, February, ..., December.
44390	`%c`	Equivalent to `%a %b %e %T %Y`.
44391	`%p`	One of AM or PM.
44392	`%r`	Equivalent to `%I:%M:%S %p`.
44393	`%x`	Equivalent to `%m/%d/%y`.
44394	`%X`	Equivalent to `%T`.
44395	`%Z`	Implementation-defined.

44396 **RATIONALE**

44397 None.

44398 **FUTURE DIRECTIONS**

44399 None.

44400 **SEE ALSO**

44401 *asctime*( ), *clock*( ), *ctime*( ), *difftime*( ), *getdate*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strptime*( ), *time*( ),
44402 *tzset*( ), *utime*( ), Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5, LC_TIME,
44403 **<time.h>**

44404 **CHANGE HISTORY**

44405 First released in Issue 3.

44406 **Issue 5**

44407 The description of `%OV` is changed to be consistent with `%V` and defines Monday as the first day
44408 of the week.

44409 The description of `%Oy` is clarified.

44410 **Issue 6**

44411 Extensions beyond the ISO C standard are marked.

44412 The Open Group Corrigendum U033/8 is applied. The `%V` conversion specifier is changed from
44413 ''Otherwise, it is week 53 of the previous year, and the next week is week 1'' to ''Otherwise, it is
44414 the last week of the previous year, and the next week is week 1''.

44415 The following new requirements on POSIX implementations derive from alignment with the
44416 Single UNIX Specification:

44417 - The `%C`, `%D`, `%e`, `%h`, `%n`, `%r`, `%R`, `%t`, and `%T` conversion specifiers are added.

44418 - The modified conversion specifiers are added for consistency with the ISO POSIX-2 standard
44419 *date* utility.

44420 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

44421 - The *strftime*( ) prototype is updated.

44422 - The DESCRIPTION is extensively revised.

44423 - The `%z` conversion specifier is added.

44424 A new example is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

44425 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied. |

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strlen — get string length

SYNOPSIS

```
#include <string.h>

size_t strlen(const char *s);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strlen*( ) function shall compute the number of bytes in the string to which *s* points, not including the terminating null byte.

RETURN VALUE

The *strlen*( ) function shall return the length of *s*; no return value shall be reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

Getting String Lengths

The following example sets the maximum length of *key* and *data* by using *strlen*( ) to get the lengths of those strings.

```
#include <string.h>
...
struct element {
    char *key;
    char *data;
};
...
char *key, *data;
int len;

*keylength = *datalength = 0;
...
if ((len = strlen(key)) > *keylength)
    *keylength = len;
if ((len = strlen(data)) > *datalength)
    *datalength = len;
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The RETURN VALUE section is updated to indicate that *strlen*( ) returns the length of *s*, and not *s* itself as was previously stated.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strncasecmp — case-insensitive string comparison

SYNOPSIS

XSI

```
#include <strings.h>

int strncasecmp(const char *s1, const char *s2, size_t n);
```

DESCRIPTION

Refer to *strcasecmp*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

strncat — concatenate a string with part of another

SYNOPSIS

```
#include <string.h>

char *strncat(char *restrict s1, const char *restrict s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strncat*( ) function shall append not more than *n* bytes (a null byte and bytes that follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites the null byte at the end of *s1*. A terminating null byte is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *strncat*( ) function shall return *s1*; no return value shall be reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcat( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *strncat*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strncmp — compare part of two strings

SYNOPSIS

```
#include <string.h>

int strncmp(const char *s1, const char *s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strncmp*( ) function shall compare not more than *n* bytes (bytes that follow a null byte are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

RETURN VALUE

Upon successful completion, *strncmp*( ) shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *s2* respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

strncpy — copy part of a string

SYNOPSIS

```
#include <string.h>

char *strncpy(char *restrict s1, const char *restrict s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strncpy*( ) function shall copy not more than *n* bytes (bytes that follow a null byte are not copied) from the array pointed to by *s2* to the array pointed to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

If the array pointed to by *s2* is a string that is shorter than *n* bytes, null bytes shall be appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

RETURN VALUE

The *strncpy*( ) function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

If there is no null byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-terminated.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcpy( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The *strncpy*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strpbrk — scan a string for a byte

SYNOPSIS

```
#include <string.h>

char *strpbrk(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strpbrk*( ) function shall locate the first occurrence in the string pointed to by *s1* of any byte from the string pointed to by *s2*.

RETURN VALUE

Upon successful completion, *strpbrk*( ) shall return a pointer to the byte or a null pointer if no byte from *s2* occurs in *s1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( ), *strrchr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

strptime — date and time conversion

SYNOPSIS

XSI
```
#include <time.h>

char *strptime(const char *restrict buf, const char *restrict format,
    struct tm *restrict tm);
```

DESCRIPTION

The *strptime*( ) function shall convert the character string pointed to by *buf* to values which are stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

The *format* is composed of zero or more directives. Each directive is composed of one of the following: one or more white-space characters (as specified by *isspace*( )); an ordinary character (neither '%' nor a white-space character); or a conversion specification. Each conversion specification is composed of a '%' character followed by a conversion character which specifies the replacement required. The application shall ensure that there is white-space or other non-alphanumeric characters between any two conversion specifications. The following conversion specifications are supported:

- %a The day of the week, using the locale's weekday names; either the abbreviated or full name may be specified.
- %A Equivalent to %a.
- %b The month, using the locale's month names; either the abbreviated or full name may be specified.
- %B Equivalent to %b.
- %c Replaced by the locale's appropriate date and time representation.
- %C The century number [00,99]; leading zeros are permitted but not required.
- %d The day of the month [01,31]; leading zeros are permitted but not required.
- %D The date as %m/%d/%y.
- %e Equivalent to %d.
- %h Equivalent to %b.
- %H The hour (24-hour clock) [00,23]; leading zeros are permitted but not required.
- %I The hour (12-hour clock) [01,12]; leading zeros are permitted but not required.
- %j The day number of the year [001,366]; leading zeros are permitted but not required.
- %m The month number [01,12]; leading zeros are permitted but not required.
- %M The minute [00,59]; leading zeros are permitted but not required.
- %n Any white space.
- %p The locale's equivalent of a.m or p.m.
- %r 12-hour clock time using the AM/PM notation if **t_fmt_ampm** is not an empty string in the LC_TIME portion of the current locale; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
- %R The time as %H:%M.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`%S` The seconds [00,60]; leading zeros are permitted but not required.

`%t` Any white space.

`%T` The time as `%H:%M:%S`.

`%U` The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.

`%w` The weekday as a decimal number [0,6], with 0 representing Sunday; leading zeros are permitted but not required.

`%W` The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.

`%x` The date, using the locale's date format.

`%X` The time, using the locale's time format.

`%y` The year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required.

Note: It is expected that in a future version of IEEE Std 1003.1-2001 the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)

`%Y` The year, including the century (for example, 1988).

`%%` Replaced by `%`.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the `E` and `O` modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist in the current locale, the behavior shall be as if the unmodified conversion specification were used.

`%Ec` The locale's alternative appropriate date and time representation.

`%EC` The name of the base year (period) in the locale's alternative representation.

`%Ex` The locale's alternative date representation.

`%EX` The locale's alternative time representation.

`%Ey` The offset from `%EC` (year only) in the locale's alternative representation.

`%EY` The full alternative year representation.

`%Od` The day of the month using the locale's alternative numeric symbols; leading zeros are permitted but not required.

`%Oe` Equivalent to `%Od`.

`%OH` The hour (24-hour clock) using the locale's alternative numeric symbols.

`%OI` The hour (12-hour clock) using the locale's alternative numeric symbols.

`%Om` The month using the locale's alternative numeric symbols.

`%OM` The minutes using the locale's alternative numeric symbols.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

%OS The seconds using the locale's alternative numeric symbols.

%OU The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.

%Ow The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.

%OW The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

%Oy The year (offset from %C) using the locale's alternative numeric symbols.

A conversion specification composed of white-space characters is executed by scanning input up to the first character that is not white-space (which remains unscanned), or until no more characters can be scanned.

A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unscanned.

A series of conversion specifications composed of %n, %t, white-space characters, or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.

Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate **tm** structure members are set to values corresponding to the locale information. Case is ignored when matching items in *buf* such as month or weekday names. If no match is found, *strptime*( ) fails and no more characters are scanned.

RETURN VALUE

Upon successful completion, *strptime*( ) shall return a pointer to the character following the last character parsed. Otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Several ''equivalent to'' formats and the special processing of white-space characters are provided in order to ease the use of identical *format* strings for *strftime*( ) and *strptime*( ).

Applications should use %Y (4-digit years) in preference to %y (2-digit years).

It is unspecified whether multiple calls to *strptime*( ) using the same **tm** structure will update the current contents of the structure or overwrite all contents of the structure. Conforming applications should make a single call to *strptime*( ) with a format and all data needed to completely specify the date and time being converted.

RATIONALE

None.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FUTURE DIRECTIONS

The *strptime*( ) function is expected to be mandatory in the next version of this volume of IEEE Std 1003.1-2001.

SEE ALSO

scanf( ), *strftime*( ), *time*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Moved from ENHANCED I18N to BASE.

The [ENOSYS] error is removed.

The exact meaning of the `%y` and `%Oy` specifiers is clarified in the DESCRIPTION.

Issue 6

The Open Group Corrigendum U033/5 is applied. The `%r` specifier description is reworded.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *strptime*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

The Open Group Corrigendum U047/2 is applied.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ for consistency with *strftime*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

-`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

strrchr — string scanning operation

SYNOPSIS

```
#include <string.h>

char *strrchr(const char *s, int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strrchr*( ) function shall locate the last occurrence of *c* (converted to a **char**) in the string pointed to by *s*. The terminating null byte is considered to be part of the string.

RETURN VALUE

Upon successful completion, *strrchr*( ) shall return a pointer to the byte or a null pointer if *c* does not occur in the string.

ERRORS

No errors are defined.

EXAMPLES

Finding the Base Name of a File

The following example uses *strrchr*( ) to get a pointer to the base name of a file. The *strrchr*( ) function searches backwards through the name of the file to find the last '/' character in *name*. This pointer (plus one) will point to the base name of the file.

```
#include <string.h>
...
const char *name;
char *basename;
...
basename = strrchr(name, '/') + 1;
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strspn — get length of a substring

SYNOPSIS

```
#include <string.h>

size_t strspn(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strspn*( ) function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

RETURN VALUE

The *strspn*( ) function shall return the length of *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcspn( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The RETURN VALUE section is updated to indicate that *strspn*( ) returns the length of *s*, and not *s* itself as was previously stated.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strstr — find a substring

SYNOPSIS

```
#include <string.h>

char *strstr(const char *s1, const char *s2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strstr*( ) function shall locate the first occurrence in the string pointed to by *s1* of the sequence of bytes (excluding the terminating null byte) in the string pointed to by *s2*.

RETURN VALUE

Upon successful completion, *strstr*( ) shall return a pointer to the located string or a null pointer if the string is not found.

If *s2* points to a string with zero length, the function shall return *s1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the ANSI C standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strtod, strtof, strtold — convert a string to a double-precision number

SYNOPSIS

```
#include <stdlib.h>

double strtod(const char *restrict nptr, char **restrict endptr);
float strtof(const char *restrict nptr, char **restrict endptr);
long double strtold(const char *restrict nptr, char **restrict endptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace*( ))
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final string of one or more unrecognized characters, including the terminating null byte of the input string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character, then an optional exponent part
- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character, then an optional binary exponent part
- One of INF or INFINITY, ignoring case
- One of NAN or NAN(*n-char-sequence*$_{opt}$), ignoring case in the NAN part, where:

```
n-char-sequence:
     digit
     nondigit
     n-char-sequence digit
     n-char-sequence nondigit
```

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) shall be interpreted as a floating constant of the C language, except that the radix character shall be used in place of a period, and that if neither an exponent part nor a radix character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

infinity, if representable in the return type, else as if it were a floating constant that is too large for the range of the return type. A character sequence NAN or NAN(*n-char-sequence*$_{opt}$) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence part that does not have the expected form; the meaning of the *n*-char sequences is implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value resulting from the conversion is correctly rounded.

CX The radix character is defined in the program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period ('.').

CX In other than the C or POSIX locales, other implementation-defined subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

CX The *strtod*( ) function shall not change the setting of *errno* if successful.

Since 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *strtod*( ), *strtof*( ), or *strtold*( ), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value. If no conversion could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

If the correct value is outside the range of representable values, ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to [ERANGE].

If the correct value would cause an underflow, a value whose magnitude is no greater than the smallest normalized positive number in the return type shall be returned and *errno* set to [ERANGE].

ERRORS

These functions shall fail if:

CX [ERANGE] The value to be returned would cause overflow or underflow.

These functions may fail if:

CX [EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in **<float.h>**) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

values of *L*, *D*, and *U* satisfy *L* <= *D* <= *U*. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

The changes to *strtod*( ) introduced by the ISO / IEC 9899: 1999 standard can alter the behavior of well-formed applications complying with the ISO / IEC 9899: 1990 standard and thus earlier versions of the base documents. One such example would be:

```
int
what_kind_of_number (char *s)
{
    char *endp;
    double d;
    long l;

    d = strtod(s, &endp);
    if (s != endp && *endp == '\0')
        printf("It's a float with value %g\n", d);
    else
    {
        l = strtol(s, &endp, 0);
        if (s != endp && *endp == '\0')
            printf("It's an integer with value %ld\n", 1);
        else
            return 1;
    }
    return 0;
}
```

If the function is called with:

```
what_kind_of_number ("0x10")
```

an ISO / IEC 9899: 1990 standard-compliant library will result in the function printing:

```
It's an integer with value 16
```

With the ISO / IEC 9899: 1999 standard, the result is:

```
It's a float with value 16
```

The change in behavior is due to the inclusion of floating-point numbers in hexadecimal notation without requiring that either a decimal point or the binary exponent be present.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isspace( ), *localeconv*( ), *scanf*( ), *setlocale*( ), *strtol*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<float.h>**, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *strtod*() function is updated.
- The *strtof*() and *strtold*() functions are added.
- The DESCRIPTION is extensively revised.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second paragraph in the RETURN VALUE section. This change clarifies the sign of the return value.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strtoimax, strtoumax — convert string to integer type

SYNOPSIS

```
#include <inttypes.h>

intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,
    int base);
uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
    int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall be equivalent to the *strtol*( ), *strtoll*( ), *strtoul*( ), and *strtoull*( ) functions, except that the initial portion of the string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

If no conversion could be performed, zero shall be returned.

If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to [ERANGE].

ERRORS

These functions shall fail if:

[ERANGE] The value to be returned is not representable.

These functions may fail if:

[EINVAL] The value of *base* is not supported.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( ), *strtoul*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<inttypes.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strtok, strtok_r — split string into tokens

SYNOPSIS

```
#include <string.h>

char *strtok(char *restrict s1, const char *restrict s2);
TSF char *strtok_r(char *restrict s, const char *restrict sep,
        char **restrict lasts);
```

DESCRIPTION

CX For *strtok*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

A sequence of calls to *strtok*() breaks the string pointed to by *s1* into a sequence of tokens, each of which is delimited by a byte from the string pointed to by *s2*. The first call in the sequence has *s1* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by *s2* may be different from call to call.

The first call in the sequence searches the string pointed to by *s1* for the first byte that is *not* contained in the current separator string pointed to by *s2*. If no such byte is found, then there are no tokens in the string pointed to by *s1* and *strtok*() shall return a null pointer. If such a byte is found, it is the start of the first token.

The *strtok*() function then searches from there for a byte that *is* contained in the current separator string. If no such byte is found, the current token extends to the end of the string pointed to by *s1*, and subsequent searches for a token shall return a null pointer. If such a byte is found, it is overwritten by a null byte, which terminates the current token. The *strtok*() function saves a pointer to the following byte, from which the next search for a token shall start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *strtok*().

CX The *strtok*() function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

TSF The *strtok_r*() function considers the null-terminated string *s* as a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *sep*. The argument *lasts* points to a user-provided pointer which points to stored information necessary for *strtok_r*() to continue scanning the same string.

In the first call to *strtok_r*(), *s* points to a null-terminated string, *sep* to a null-terminated string of separator characters, and the value pointed to by *lasts* is ignored. The *strtok_r*() function shall return a pointer to the first character of the first token, write a null character into *s* immediately following the returned token, and update the pointer to which *lasts* points.

In subsequent calls, *s* is a NULL pointer and *lasts* shall be unchanged from the previous call so that subsequent calls shall move through the string *s*, returning successive tokens until no tokens remain. The separator string *sep* may be different from call to call. When no token remains in *s*, a NULL pointer shall be returned.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RETURN VALUE

Upon successful completion, *strtok*( ) shall return a pointer to the first byte of a token. Otherwise, if there is no token, *strtok*( ) shall return a null pointer.

TSF The *strtok_r*( ) function shall return a pointer to the token found, or a NULL pointer when no token is found.

ERRORS

No errors are defined.

EXAMPLES

Searching for Word Separators

The following example searches for tokens separated by <space>s.

```
#include <string.h>
...
char *token;
char *line = "LINE TO BE SEPARATED";
char *search = " ";

/* Token will point to "LINE". */
token = strtok(line, search);

/* Token will point to "TO". */
token = strtok(NULL, search);
```

Breaking a Line

The following example uses *strtok*( ) to break a line into two character strings separated by any combination of <space>s, <tab>s, or <newline>s.

```
#include <string.h>
...
struct element {
    char *key;
    char *data;
};
...
char line[LINE_MAX];
char *key, *data;
...
key = strtok(line, " \n");
data = strtok(NULL, " \n");
...
```

APPLICATION USAGE

The *strtok_r*( ) function is thread-safe and stores its state in a user-supplied buffer instead of possibly using a static data area that may be overwritten by an unrelated call from another thread.

RATIONALE

The *strtok*( ) function searches for a separator string within a larger string. It returns a pointer to the last substring between separator strings. This function uses static storage to keep track of the current string position between calls. The new function, *strtok_r*( ), takes an additional argument, *lasts*, to keep track of the current position in the string.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *strtok_r*( ) function is included for alignment with the POSIX Threads Extension.

A note indicating that the *strtok*( ) function need not be reentrant is added to the DESCRIPTION.

Issue 6

Extensions beyond the ISO C standard are marked.

The *strtok_r*( ) function is marked as part of the Thread-Safe Functions option.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *strtok*( ) and *strtok_r*( ) prototypes for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

strtol, strtoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>

long strtol(const char *restrict str, char **restrict endptr, int base);
long long strtoll(const char *restrict str, char **restrict endptr,
    int base)
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and **long long** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace*( ))
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final string of one or more unrecognized characters, including the terminating null byte of the input string.

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '−' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a '+' or '−' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

CX In other than the C or POSIX locales, other implementation-defined subject sequences may be accepted.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

CX The *strtol*( ) function shall not change the setting of *errno* if successful.

Since 0, {LONG_MIN} or {LLONG_MIN}, and {LONG_MAX} or {LLONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set *errno* to 0, then call *strtol*( ) or *strtoll*( ), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value, if any. If no CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX}, {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and *errno* set to [ERANGE].

ERRORS

These functions shall fail if:

[ERANGE] The value to be returned is not representable.

These functions may fail if:

CX [EINVAL] The value of *base* is not supported.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalpha( ), *scanf*( ), *strtod*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO / IEC 9899: 1999 standard:

- The *strtol*( ) prototype is updated.
- The *strtoll*( ) function is added.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45221 **NAME**

45222 strtold — convert a string to a double-precision number

45223 **SYNOPSIS**

45224 `#include <stdlib.h>`

45225 `long double strtold(const char *restrict` *`nptr`*`, char **restrict` *`endptr`*`);`

45226 **DESCRIPTION**

45227 Refer to *strtod*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45228 **NAME**

45229 strtoll — convert a string to a long integer

45230 **SYNOPSIS**

45231 `#include <stdlib.h>`

45232 `long long strtoll(const char *restrict` *`str`*`, char **restrict` *`endptr`*`,`
45233 `int` *`base`*`);`

45234 **DESCRIPTION**

45235 Refer to *strtol*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strtoul, strtoull — convert a string to an unsigned long

SYNOPSIS

```
#include <stdlib.h>

unsigned long strtoul(const char *restrict str,
    char **restrict endptr, int base);
unsigned long long strtoull(const char *restrict str,
    char **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned long** and **unsigned long long** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace*( ))
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final string of one or more unrecognized characters, including the terminating null byte of the input string

Then they shall attempt to convert the subject sequence to an unsigned integer, and return the result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '−' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a '+' or '−' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45281 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
45282 accepted.

45283 If the subject sequence is empty or does not have the expected form, no conversion shall be
45284 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*
45285 is not a null pointer.

45286 CX The *strtoul*( ) function shall not change the setting of *errno* if successful.

45287 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and are also valid returns
45288 on success, an application wishing to check for error situations should set *errno* to 0, then call
45289 *strtoul*( ) or *strtoull*( ), then check *errno*.

45290 **RETURN VALUE**

45291 Upon successful completion, these functions shall return the converted value, if any. If no
45292 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the
45293 correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX}
45294 shall be returned and *errno* set to [ERANGE].

45295 **ERRORS**

45296 These functions shall fail if:

45297 CX [EINVAL] The value of *base* is not supported.

45298 [ERANGE] The value to be returned is not representable.

45299 These functions may fail if:

45300 CX [EINVAL] No conversion could be performed.

45301 **EXAMPLES**

45302 None.

45303 **APPLICATION USAGE**

45304 None.

45305 **RATIONALE**

45306 None.

45307 **FUTURE DIRECTIONS**

45308 None.

45309 **SEE ALSO**

45310 *isalpha*( ), *scanf*( ), *strtod*( ), *strtol*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
45311 **<stdlib.h>**

45312 **CHANGE HISTORY**

45313 First released in Issue 4. Derived from the ANSI C standard.

45314 **Issue 5**

45315 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

45316 **Issue 6**

45317 Extensions beyond the ISO C standard are marked.

45318 The following new requirements on POSIX implementations derive from alignment with the
45319 Single UNIX Specification:

45320 - The [EINVAL] error condition is added for when the value of *base* is not supported.

45321 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
45322 added if no conversion could be performed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The following changes are made for alignment with the ISO / IEC 9899: 1999 standard:

- The *strtoul*( ) prototype is updated.
- The *strtoull*( ) function is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45326 **NAME**

45327 strtoumax — convert a string to an integer type

45328 **SYNOPSIS**

45329
```
#include <inttypes.h>
```

45330
```
uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
```
45331
```
    int base);
```

45332 **DESCRIPTION**

45333 Refer to *strtoimax*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

strxfrm — string transformation

SYNOPSIS

```
#include <string.h>

size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *strxfrm*( ) function shall transform the string pointed to by *s2* and place the resulting string into the array pointed to by *s1*. The transformation is such that if *strcmp*( ) is applied to two transformed strings, it shall return a value greater than, equal to, or less than 0, corresponding to the result of *strcoll*( ) applied to the same two original strings. No more than *n* bytes are placed into the resulting array pointed to by *s1*, including the terminating null byte. If *n* is 0, *s1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

CX The *strxfrm*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strxfrm*( ), then check *errno*.

RETURN VALUE

Upon successful completion, *strxfrm*( ) shall return the length of the transformed string (not including the terminating null byte). If the value returned is *n* or more, the contents of the array pointed to by *s1* are unspecified.

CX On error, *strxfrm*( ) may set *errno* but no return value is reserved to indicate an error.

ERRORS

The *strxfrm*( ) function may fail if:

CX [EINVAL] The string pointed to by the *s2* argument contains characters outside the domain of the collating sequence.

EXAMPLES

None.

APPLICATION USAGE

The transformation function is such that two transformed strings can be ordered by *strcmp*( ) as appropriate to collating sequence information in the program's locale (category *LC_COLLATE*).

The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the *s1* array prior to making the transformation.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp( ), *strcoll*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45375 **CHANGE HISTORY**

45376 First released in Issue 3. Included for alignment with the ISO C standard.

45377 **Issue 5**

45378 The DESCRIPTION is updated to indicate that *errno* does not change if the function is
45379 successful.

45380 **Issue 6**

45381 Extensions beyond the ISO C standard are marked.

45382 The following new requirements on POSIX implementations derive from alignment with the
45383 Single UNIX Specification:

45384 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
45385 added if no conversion could be performed.

45386 The *strxfrm*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

swab — swap bytes

SYNOPSIS

XSI
```
#include <unistd.h>

void swab(const void *restrict src, void *restrict dest,
    ssize_t nbytes);
```

DESCRIPTION

The *swab*( ) function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab*( ) copies and exchanges *nbytes*−1 bytes and the disposition of the last byte is unspecified. If copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is negative, *swab*( ) does nothing.

RETURN VALUE

None.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The **restrict** keyword is added to the *swab*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45419 **NAME**

45420 swapcontext — swap user context

45421 **SYNOPSIS**

45422 XSI
```
#include <ucontext.h>

int swapcontext(ucontext_t *restrict oucp,
    const ucontext_t *restrict ucp);
```
45423
45424
45425

45426 **DESCRIPTION**

45427 Refer to *makecontext*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

45428 **NAME**

45429 swprintf — print formatted wide-character output

45430 **SYNOPSIS**

45431 `#include <stdio.h>`

45432 `#include <wchar.h>`

45433 `int swprintf(wchar_t *restrict ws, size_t n,`

45434 `const wchar_t *restrict format, ...);`

45435 **DESCRIPTION**

45436 Refer to *fwprintf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45437 **NAME**

45438 swscanf — convert formatted wide-character input

45439 **SYNOPSIS**

45440 `#include <stdio.h>`

45441 `#include <wchar.h>`

45442 `int swscanf(const wchar_t *restrict ws,`

45443 `const wchar_t *restrict format, ... );`

45444 **DESCRIPTION**

45445 Refer to *fwscanf*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

45446 **NAME**

45447 symlink — make a symbolic link to a file

45448 **SYNOPSIS**

45449 `#include <unistd.h>`

45450 `int symlink(const char *path1, const char *path2);`

45451 **DESCRIPTION**

45452 The *symlink*( ) function shall create a symbolic link called *path2* that contains the string pointed
45453 to by *path1* (*path2* is the name of the symbolic link created, *path1* is the string contained in the
45454 symbolic link).

45455 The string pointed to by *path1* shall be treated only as a character string and shall not be
45456 validated as a pathname.

45457 If the *symlink*( ) function fails for any reason other than [EIO], any file named by *path2* shall be
45458 unaffected.

45459 **RETURN VALUE**

45460 Upon successful completion, *symlink*( ) shall return 0; otherwise, it shall return –1 and set *errno* to
45461 indicate the error.

45462 **ERRORS**

45463 The *symlink*( ) function shall fail if:

45464 [EACCES] Write permission is denied in the directory where the symbolic link is being
45465 created, or search permission is denied for a component of the path prefix of
45466 *path2*.

45467 [EEXIST] The *path2* argument names an existing file or symbolic link.

45468 [EIO] An I/O error occurs while reading from or writing to the file system.

45469 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path2*
45470 argument.

45471 [ENAMETOOLONG]
45472 The length of the *path2* argument exceeds {PATH_MAX} or a pathname
45473 component is longer than {NAME_MAX} or the length of the *path1* argument
45474 is longer than {SYMLINK_MAX}.

45475 [ENOENT] A component of *path2* does not name an existing file or *path2* is an empty
45476 string.

45477 [ENOSPC] The directory in which the entry for the new symbolic link is being placed
45478 cannot be extended because no space is left on the file system containing the
45479 directory, or the new symbolic link cannot be created because no space is left
45480 on the file system which shall contain the link, or the file system is out of file-
45481 allocation resources.

45482 [ENOTDIR] A component of the path prefix of *path2* is not a directory.

45483 [EROFS] The new symbolic link would reside on a read-only file system.

45484 The *symlink*( ) function may fail if:

45485 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
45486 resolution of the *path2* argument.

45487 [ENAMETOOLONG]
45488 As a result of encountering a symbolic link in resolution of the *path2*

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

argument, the length of the substituted pathname string exceeded {PATH_MAX} bytes (including the terminating null byte), or the length of the string pointed to by *path1* exceeded {SYMLINK_MAX}.

EXAMPLES

None.

APPLICATION USAGE

Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not exist when the link is created. A symbolic link can cross file system boundaries.

Normal permission checks are made on each component of the symbolic link pathname during its resolution.

RATIONALE

Since IEEE Std 1003.1-2001 does not require any association of file times with symbolic links, there is no requirement that file times be updated by *symlink*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

lchown( ), *link*( ), *lstat*( ), *open*( ), *readlink*( ), *unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION text is updated.
- The [ELOOP] optional error condition is added.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45517 **NAME**

45518 sync — schedule file system updates

45519 **SYNOPSIS**

45520 XSI
```
#include <unistd.h>
```

45521
```
void sync(void);
```

45522

45523 **DESCRIPTION**

45524 The *sync*( ) function shall cause all information in memory that updates file systems to be
45525 scheduled for writing out to all file systems.

45526 The writing, although scheduled, is not necessarily complete upon return from *sync*( ).

45527 **RETURN VALUE**

45528 The *sync*( ) function shall not return a value.

45529 **ERRORS**

45530 No errors are defined.

45531 **EXAMPLES**

45532 None.

45533 **APPLICATION USAGE**

45534 None.

45535 **RATIONALE**

45536 None.

45537 **FUTURE DIRECTIONS**

45538 None.

45539 **SEE ALSO**

45540 *fsync*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

45541 **CHANGE HISTORY**

45542 First released in Issue 4, Version 2.

45543 **Issue 5**

45544 Moved from X/OPEN UNIX extension to BASE.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

sysconf — get configurable system variables

SYNOPSIS

```
#include <unistd.h>

long sysconf(int name);
```

DESCRIPTION

The *sysconf*( ) function provides a method for the application to determine the current value of a configurable system limit or option (*variable*). The implementation shall support all of the variables listed in the following table and may support others.

The *name* argument represents the system variable to be queried. The following table lists the minimal set of system variables from **<limits.h>** or **<unistd.h>** that can be returned by *sysconf*( ), and the symbolic constants defined in **<unistd.h>** that are the corresponding values used for *name*.

Variable	Value of Name
{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
{AIO_MAX}	_SC_AIO_MAX
{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
{ARG_MAX}	_SC_ARG_MAX
{ATEXIT_MAX}	_SC_ATEXIT_MAX
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{CHILD_MAX}	_SC_CHILD_MAX
Clock ticks/second	_SC_CLK_TCK
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
{IOV_MAX}	_SC_IOV_MAX
{LINE_MAX}	_SC_LINE_MAX
{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
{NGROUPS_MAX}	_SC_NGROUPS_MAX
Maximum size of *getgrgid_r*( ) and *getgrnam_r*( ) data buffers	_SC_GETGR_R_SIZE_MAX
Maximum size of *getpwuid_r*( ) and *getpwnam_r*( ) data buffers	_SC_GETPW_R_SIZE_MAX
{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
{OPEN_MAX}	_SC_OPEN_MAX
_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
_POSIX_BARRIERS	_SC_BARRIERS
_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
_POSIX_CPUTIME	_SC_CPUTIME

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Variable	Value of Name
_POSIX_FILE_LOCKING	_SC_FILE_LOCKING
_POSIX_FSYNC	_SC_FSYNC
_POSIX_IPV6	_SC_IPV6
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL
_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
_POSIX_MEMLOCK	_SC_MEMLOCK
_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
_POSIX_MULTI_PROCESS	_SC_MULTI_PROCESS
_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
_POSIX_REGEXP	_SC_REGEXP
_POSIX_SAVED_IDS	_SC_SAVED_IDS
_POSIX_SEMAPHORES	_SC_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
_POSIX_SHELL	_SC_SHELL
_POSIX_SPAWN	_SC_SPAWN
_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
_POSIX_SYMLOOP_MAX	_SC_SYMLOOP_MAX
_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_CPUTIME	_SC_THREAD_CPUTIME
_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
_POSIX_THREAD_SPORADIC_SERVER	_SC_THREAD_SPORADIC_SERVER
_POSIX_THREADS	_SC_THREADS
_POSIX_TIMEOUTS	_SC_TIMEOUTS
_POSIX_TIMERS	_SC_TIMERS
_POSIX_TRACE	_SC_TRACE
_POSIX_TRACE_EVENT_FILTER	_SC_TRACE_EVENT_FILTER
_POSIX_TRACE_INHERIT	_SC_TRACE_INHERIT
_POSIX_TRACE_LOG	_SC_TRACE_LOG
_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
_POSIX_VERSION	_SC_VERSION
_POSIX_V6_ILP32_OFF32	_SC_V6_ILP32_OFF32
_POSIX_V6_ILP32_OFFBIG	_SC_V6_ILP32_OFFBIG
_POSIX_V6_LP64_OFF64	_SC_V6_LP64_OFF64

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Variable	Value of Name
_POSIX_V6_LPBIG_OFFBIG	_SC_V6_LPBIG_OFFBIG
_POSIX2_C_BIND	_SC_2_C_BIND
_POSIX2_C_DEV	_SC_2_C_DEV
_POSIX2_C_VERSION	_SC_2_C_VERSION
_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
_POSIX2_FORT_DEV	_SC_2_FORT_DEV
_POSIX2_FORT_RUN	_SC_2_FORT_RUN
_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
_POSIX2_PBS	_SC_2_PBS
_POSIX2_PBS_ACCOUNTING	_SC_2_PBS_ACCOUNTING
_POSIX2_PBS_CHECKPOINT	_SC_2_PBS_CHECKPOINT
_POSIX2_PBS_LOCATE	_SC_2_PBS_LOCATE
_POSIX2_PBS_MESSAGE	_SC_2_PBS_MESSAGE
_POSIX2_PBS_TRACK	_SC_2_PBS_TRACK
_POSIX2_SW_DEV	_SC_2_SW_DEV
_POSIX2_UPE	_SC_2_UPE
_POSIX2_VERSION	_SC_2_VERSION
_REGEX_VERSION	_SC_REGEX_VERSION
{PAGE_SIZE}	_SC_PAGE_SIZE
{PAGESIZE}	_SC_PAGESIZE
{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS
{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
{RE_DUP_MAX}	_SC_RE_DUP_MAX
{RTSIG_MAX}	_SC_RTSIG_MAX
{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
{STREAM_MAX}	_SC_STREAM_MAX
{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
{TIMER_MAX}	_SC_TIMER_MAX
{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
{TZNAME_MAX}	_SC_TZNAME_MAX
_XBS5_ILP32_OFF32 (**LEGACY**)	_SC_XBS5_ILP32_OFF32 (**LEGACY**)
_XBS5_ILP32_OFFBIG (**LEGACY**)	_SC_XBS5_ILP32_OFFBIG (**LEGACY**)
_XBS5_LP64_OFF64 (**LEGACY**)	_SC_XBS5_LP64_OFF64 (**LEGACY**)
_XBS5_LPBIG_OFFBIG (**LEGACY**)	_SC_XBS5_LPBIG_OFFBIG (**LEGACY**)
_XOPEN_CRYPT	_SC_XOPEN_CRYPT
_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
_XOPEN_LEGACY	_SC_XOPEN_LEGACY
_XOPEN_REALTIME	_SC_XOPEN_REALTIME
_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
_XOPEN_SHM	_SC_XOPEN_SHM
_XOPEN_STREAMS	_SC_XOPEN_STREAMS
_XOPEN_UNIX	_SC_XOPEN_UNIX
_XOPEN_VERSION	_SC_XOPEN_VERSION

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Variable	Value of Name
_XOPEN_XCU_VERSION	_SC_XOPEN_XCU_VERSION

RETURN VALUE

If *name* is an invalid value, *sysconf*( ) shall return –1 and set *errno* to indicate the error. If the variable corresponding to *name* has no limit, *sysconf*( ) shall return –1 without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

Otherwise, *sysconf*( ) shall return the current variable value on the system. The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's **<limits.h>** or **<unistd.h>**. The value shall not change during the lifetime of the calling process, except that *sysconf*(_SC_OPEN_MAX) may return different values before and after a call to *setrlimit*( ) which changes the RLIMIT_NOFILE soft limit.

XSI

ERRORS

The *sysconf*( ) function shall fail if:

[EINVAL] The value of the *name* argument is invalid.

EXAMPLES

None.

APPLICATION USAGE

As –1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *sysconf*( ), and, if it returns –1, check to see if *errno* is non-zero.

If the value of *sysconf*(_SC_2_VERSION) is not equal to the value of the _POSIX2_VERSION symbolic constant, the utilities available via *system*( ) or *popen*( ) might not behave as described in the Shell and Utilities volume of IEEE Std 1003.1-2001. This would mean that the application is not running in an environment that conforms to the Shell and Utilities volume of IEEE Std 1003.1-2001. Some applications might be able to deal with this, others might not. However, the functions defined in this volume of IEEE Std 1003.1-2001 continue to operate as specified, even if *sysconf*(_SC_2_VERSION) reports that the utilities no longer perform as specified.

RATIONALE

This functionality was added in response to requirements of application developers and of system vendors who deal with many international system configurations. It is closely related to *pathconf*( ) and *fpathconf*( ).

Although a conforming application can run on all systems by never demanding more resources than the minimum values published in this volume of IEEE Std 1003.1-2001, it is useful for that application to be able to use the actual value for the quantity of a resource available on any given system. To do this, the application makes use of the value of a symbolic constant in **<limits.h>** or **<unistd.h>**.

However, once compiled, the application must still be able to cope if the amount of resource available is increased. To that end, an application may need a means of determining the quantity of a resource, or the presence of an option, at execution time.

Two examples are offered:

1. Applications may wish to act differently on systems with or without job control. Applications vendors who wish to distribute only a single binary package to all instances of a computer architecture would be forced to assume job control is never available if it

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

were to rely solely on the **<unistd.h>** value published in this volume of IEEE Std 1003.1-2001.

2. International applications vendors occasionally require knowledge of the number of clock ticks per second. Without these facilities, they would be required to either distribute their applications partially in source form or to have 50 Hz and 60 Hz versions for the various countries in which they operate.

It is the knowledge that many applications are actually distributed widely in executable form that leads to this facility. If limited to the most restrictive values in the headers, such applications would have to be prepared to accept the most limited environments offered by the smallest microcomputers. Although this is entirely portable, there was a consensus that they should be able to take advantage of the facilities offered by large systems, without the restrictions associated with source and object distributions.

During the discussions of this feature, it was pointed out that it is almost always possible for an application to discern what a value might be at runtime by suitably testing the various functions themselves. And, in any event, it could always be written to adequately deal with error returns from the various functions. In the end, it was felt that this imposed an unreasonable level of complication and sophistication on the application writer.

This runtime facility is not meant to provide ever-changing values that applications have to check multiple times. The values are seen as changing no more frequently than once per system initialization, such as by a system administrator or operator with an automatic configuration program. This volume of IEEE Std 1003.1-2001 specifies that they shall not change within the lifetime of the process.

Some values apply to the system overall and others vary at the file system or directory level. The latter are described in *pathconf*( ).

Note that all values returned must be expressible as integers. String values were considered, but the additional flexibility of this approach was rejected due to its added complexity of implementation and use.

Some values, such as {PATH_MAX}, are sometimes so large that they must not be used to, say, allocate arrays. The *sysconf*( ) function returns a negative value to show that this symbolic constant is not even defined in this case.

Similar to *pathconf*( ), this permits the implementation not to have a limit. When one resource is infinite, returning an error indicating that some other resource limit has been reached is conforming behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

confstr( ), *pathconf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<unistd.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, *getconf*

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The _XBS_ variables and name values are added to the table of system variables in the DESCRIPTION. These are all marked EX.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 6

The symbol CLK_TCK is obsolescent and removed. It is replaced with the phrase ‘‘clock ticks per second’’.

The symbol {PASS_MAX} is removed.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Table entries are added for the following variables: _SC_REGEXP, _SC_SHELL, _SC_REGEX_VERSION, _SC_SYMLOOP_MAX.

The following *sysconf*( ) variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

_POSIX_ADVISORY_INFO
_POSIX_CPUTIME
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TIMEOUTS

The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

- A statement expressing the dependency of support for some system variables on implementation options is added.
- The following system variables are added:

  _POSIX_BARRIERS
  _POSIX_CLOCK_SELECTION
  _POSIX_MONOTONIC_CLOCK
  _POSIX_READER_WRITER_LOCKS
  _POSIX_SPIN_LOCKS
  _POSIX_TYPED_MEMORY_OBJECTS

The following system variables are added for alignment with IEEE Std 1003.2d-1994:

_POSIX2_PBS
_POSIX2_PBS_ACCOUNTING
_POSIX2_PBS_LOCATE
_POSIX2_PBS_MESSAGE
_POSIX2_PBS_TRACK

The following *sysconf*( ) variables and their associated names are added for alignment with IEEE Std 1003.1q-2000:

_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_INHERIT
_POSIX_TRACE_LOG

The macros associated with the *c89* programming models are marked LEGACY, and new equivalent macros associated with *c99* are introduced.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the DESCRIPTION to denote that the _PC* and _SC* symbols are now required to be supported. A corresponding change has been made in the Base Definitions volume of IEEE Std 1003.1-2001. The deletion in the second paragraph removes some duplicated text. Additional symbols that were erroneously omitted from this reference page have been added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the RETURN VALUE section that the value returned for *sysconf*(_SC_OPEN_MAX) may change if a call to *setrlimit*() adjusts the RLIMIT_NOFILE soft limit.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

45828 **NAME**

45829 syslog — log a message

45830 **SYNOPSIS**

45831 XSI
```
#include <syslog.h>
```

45832
```
void syslog(int priority, const char *message, ... /* argument */);
```

45833

45834 **DESCRIPTION**

45835 Refer to *closelog*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

system — issue a command

SYNOPSIS

```
#include <stdlib.h>

int system(const char *command);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If *command* is a null pointer, the *system*( ) function shall determine whether the host environment has a command processor. If *command* is not a null pointer, the *system*( ) function shall pass the string pointed to by *command* to that command processor to be executed in an implementation-defined manner; this might then cause the program calling *system*( ) to behave in a non-conforming manner or to terminate.

CX The environment of the executed command shall be as if a child process were created using *fork*( ), and the child process invoked the *sh* utility using *execl*( ) as follows:

```
execl(<shell path>, "sh", "-c", command, (char *)0);
```

where <*shell path*> is an unspecified pathname for the *sh* utility.

The *system*( ) function shall ignore the SIGINT and SIGQUIT signals, and shall block the SIGCHLD signal, while waiting for the command to terminate. If this might cause the application to miss a signal that would have killed it, then the application should examine the return value from *system*( ) and take whatever action is appropriate to the application if the command terminated due to receipt of a signal.

The *system*( ) function shall not affect the termination status of any child of the calling processes other than the process or processes it itself creates.

The *system*( ) function shall not return until the child process has terminated.

RETURN VALUE

If *command* is a null pointer, *system*( ) shall return non-zero to indicate that a command processor is available, or zero if none is available. CX The *system*( ) function shall always return non-zero when *command* is NULL.

CX If *command* is not a null pointer, *system*( ) shall return the termination status of the command language interpreter in the format specified by *waitpid*( ). The termination status shall be as defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents the command language interpreter from executing after the child process is created, the return value from *system*( ) shall be as if the command language interpreter had terminated using *exit*(127) or *_exit*(127). If a child process cannot be created, or if the termination status for the command language interpreter cannot be obtained, *system*( ) shall return –1 and set *errno* to indicate the error.

ERRORS

CX The *system*( ) function may set *errno* values as described by *fork*( ).

In addition, *system*( ) may fail if:

CX [ECHILD] The status of the child process created by *system*( ) is no longer available.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

If the return value of *system*( ) is not –1, its value can be decoded through the use of the macros described in **<sys/wait.h>**. For convenience, these macros are also provided in **<stdlib.h>**.

Note that, while *system*( ) must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting for the child to terminate, the handling of signals in the executed command is as specified by *fork*( ) and *exec*. For example, if SIGINT is being caught or is set to SIG_DFL when *system*( ) is called, then the child is started with SIGINT handling set to SIG_DFL.

Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two processes reading from the same terminal, for example) when the executed command ignores or catches one of the signals. It is also usually the correct action when the user has given a command to the application to be executed synchronously (as in the `'!'` command in many interactive applications). In either case, the signal should be delivered only to the child process, not to the application itself. There is one situation where ignoring the signals might have less than the desired effect. This is when the application uses *system*( ) to perform some task invisible to the user. If the user typed the interrupt character (`"^C"`, for example) while *system*( ) is being used in this way, one would expect the application to be killed, but only the executed command is killed. Applications that use *system*( ) in this way should carefully check the return status from *system*( ) to see if the executed command was successful, and should take appropriate action when the command fails.

Blocking SIGCHLD while waiting for the child to terminate prevents the application from catching the signal and obtaining status from *system*( )'s child process before *system*( ) can get the status itself.

The context in which the utility is ultimately executed may differ from that in which *system*( ) was called. For example, file descriptors that have the FD_CLOEXEC flag set are closed, and the process ID and parent process ID are different. Also, if the executed utility changes its environment variables or its current working directory, that change is not reflected in the caller's context.

There is no defined way for an application to find the specific path for the shell. However, *confstr*( ) can provide a value for *PATH* that is guaranteed to find the *sh* utility.

RATIONALE

The *system*( ) function should not be used by programs that have set user (or group) ID privileges. The *fork*( ) and *exec* family of functions (except *execlp*( ) and *execvp*( )), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

There are three levels of specification for the *system*( ) function. The ISO C standard gives the most basic. It requires that the function exists, and defines a way for an application to query whether a command language interpreter exists. It says nothing about the command language or the environment in which the command is interpreted.

IEEE Std 1003.1-2001 places additional restrictions on *system*( ). It requires that if there is a command language interpreter, the environment must be as specified by *fork*( ) and *exec*. This ensures, for example, that close-on-*exec* works, that file locks are not inherited, and that the process ID is different. It also specifies the return value from *system*( ) when the command line can be run, thus giving the application some information about the command's completion status.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Finally, IEEE Std 1003.1-2001 requires the command to be interpreted as in the shell command language defined in the Shell and Utilities volume of IEEE Std 1003.1-2001.

Note that, *system*(NULL) is required to return non-zero, indicating that there is a command language interpreter. At first glance, this would seem to conflict with the ISO C standard which allows *system*(NULL) to return zero. There is no conflict, however. A system must have a command language interpreter, and is non-conforming if none is present. It is therefore permissible for the *system*( ) function on such a system to implement the behavior specified by the ISO C standard as long as it is understood that the implementation does not conform to IEEE Std 1003.1-2001 if *system*(NULL) returns zero.

It was explicitly decided that when *command* is NULL, *system*( ) should not be required to check to make sure that the command language interpreter actually exists with the correct mode, that there are enough processes to execute it, and so on. The call *system*(NULL) could, theoretically, check for such problems as too many existing child processes, and return zero. However, it would be inappropriate to return zero due to such a (presumably) transient condition. If some condition exists that is not under the control of this application and that would cause any *system*( ) call to fail, that system has been rendered non-conforming.

Early drafts required, or allowed, *system*( ) to return with *errno* set to [EINTR] if it was interrupted with a signal. This error return was removed, and a requirement that *system*( ) not return until the child has terminated was added. This means that if a *waitpid*( ) call in *system*( ) exits with *errno* set to [EINTR], *system*( ) must reissue the *waitpid*( ). This change was made for two reasons:

1. There is no way for an application to clean up if *system*( ) returns [EINTR], short of calling *wait*( ), and that could have the undesirable effect of returning the status of children other than the one started by *system*( ).
2. While it might require a change in some historical implementations, those implementations already have to be changed because they use *wait*( ) instead of *waitpid*( ).

Note that if the application is catching SIGCHLD signals, it will receive such a signal before a successful *system*( ) call returns.

To conform to IEEE Std 1003.1-2001, *system*( ) must use *waitpid*( ), or some similar function, instead of *wait*( ).

The following code sample illustrates how *system*( ) might be implemented on an implementation conforming to IEEE Std 1003.1-2001.

```
#include <signal.h>
int system(const char *cmd)
{
    int stat;
    pid_t pid;
    struct sigaction sa, savintr, savequit;
    sigset_t saveblock;
    if (cmd == NULL)
        return(1);
    sa.sa_handler = SIG_IGN;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigemptyset(&savintr.sa_mask);
    sigemptyset(&savequit.sa_mask);
    sigaction(SIGINT, &sa, &savintr);
    sigaction(SIGQUIT, &sa, &savequit);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
45972           sigaddset(&sa.sa_mask, SIGCHLD);
45973           sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
45974           if ((pid = fork()) == 0) {
45975               sigaction(SIGINT, &savintr, (struct sigaction *)0);
45976               sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45977               sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45978               execl("/bin/sh", "sh", "-c", cmd, (char *)0);
45979               _exit(127);
45980           }
45981           if (pid == -1) {
45982               stat = -1; /* errno comes from fork() */
45983           } else {
45984               while (waitpid(pid, &stat, 0) == -1) {
45985                   if (errno != EINTR){
45986                       stat = -1;
45987                       break;
45988                   }
45989               }
45990           }
45991           sigaction(SIGINT, &savintr, (struct sigaction *)0);
45992           sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45993           sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45994           return(stat);
45995       }
```

Note that, while a particular implementation of *system*( ) (such as the one above) can assume a particular path for the shell, such a path is not necessarily valid on another system. The above example is not portable, and is not intended to be.

One reviewer suggested that an implementation of *system*( ) might want to use an environment variable such as *SHELL* to determine which command interpreter to use. The supposed implementation would use the default command interpreter if the one specified by the environment variable was not available. This would allow a user, when using an application that prompts for command lines to be processed using *system*( ), to specify a different command interpreter. Such an implementation is discouraged. If the alternate command interpreter did not follow the command line syntax specified in the Shell and Utilities volume of IEEE Std 1003.1-2001, then changing *SHELL* would render *system*( ) non-conforming. This would affect applications that expected the specified behavior from *system*( ), and since the Shell and Utilities volume of IEEE Std 1003.1-2001 does not mention that *SHELL* affects *system*( ), the application would not know that it needed to unset *SHELL*.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *pipe*( ), *waitpid*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<signal.h>**, **<stdlib.h>**, **<sys/wait.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, *sh*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tan, tanf, tanl — tangent function

SYNOPSIS

```
#include <math.h>

double tan(double x);
float tanf(float x);
long double tanl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the tangent of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the tangent of *x*.

If the correct value would cause underflow, and is not representable, a range error may occur, and MX either 0.0 (if supported), or an implementation-defined value shall be returned.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

XSI If the correct value would cause overflow, a range error shall occur and *tan*(), *tanf*(), and *tanl*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

ERRORS

These functions shall fail if:

MX

Domain Error	The value of *x* is ±Inf.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

XSI

Range Error	The result overflows
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

These functions may fail if:

MX Range Error — The result underflows, or the value of *x* is subnormal.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

Taking the Tangent of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45.0 * M_PI / 180;
double result;
...
result = tan (radians);
```

APPLICATION USAGE

There are no known floating-point representations such that for a normal argument, *tan*(*x*) is either overflow or underflow.

These functions may lose accuracy when their argument is near a multiple of $\pi/2$ or is far from 0.0.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes in previous issues.

Issue 6

The *tanf*( ) and *tanl*( ) functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last paragraph in the RETURN VALUE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tanh, tanhf, tanhl — hyperbolic tangent functions

SYNOPSIS

```
#include <math.h>

double tanh(double x);
float tanhf(float x);
long double tanhl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the hyperbolic tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic tangent of *x*.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, ±1 shall be returned.

If *x* is subnormal, a range error may occur and *x* should be returned.

ERRORS

These functions may fail if:

MX

Range Error	The value of *x* is subnormal.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atanh( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tan*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *tanhf*( ) and *tanhl*( ) functions are added for alignment with the ISO / IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO / IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO / IEC 9899: 1999 standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

46155 **NAME**

46156 tanl — tangent function

46157 **SYNOPSIS**

46158
```
#include <math.h>
```

46159
```
long double tanl(long double x);
```

46160 **DESCRIPTION**

46161 Refer to *tan*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

tcdrain — wait for transmission of output

SYNOPSIS

```
#include <termios.h>

int tcdrain(int fildes);
```

DESCRIPTION

The *tcdrain*( ) function shall block until all output written to the object referred to by *fildes* is transmitted. The *fildes* argument is an open file descriptor associated with a terminal.

Any attempts to use *tcdrain*( ) from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcdrain*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINTR] A signal interrupted *tcdrain*( ).

[ENOTTY] The file associated with *fildes* is not a terminal.

The *tcdrain*( ) function may fail if:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcflush( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the final paragraph is no longer conditional on _POSIX_JOB_CONTROL. This is a FIPS requirement.
- The [EIO] error is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

tcflow — suspend or restart the transmission or reception of data

SYNOPSIS

```
#include <termios.h>

int tcflow(int fildes, int action);
```

DESCRIPTION

The *tcflow*( ) function shall suspend or restart transmission or reception of data on the object referred to by *fildes*, depending on the value of *action*. The *fildes* argument is an open file descriptor associated with a terminal.

- If *action* is TCOOFF, output shall be suspended.
- If *action* is TCOON, suspended output shall be restarted.
- If *action* is TCIOFF, the system shall transmit a STOP character, which is intended to cause the terminal device to stop transmitting data to the system.
- If *action* is TCION, the system shall transmit a START character, which is intended to cause the terminal device to start transmitting data to the system.

The default on the opening of a terminal file is that neither its input nor its output are suspended.

Attempts to use *tcflow*( ) from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcflow*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] The *action* argument is not a supported value.

[ENOTTY] The file associated with *fildes* is not a terminal.

The *tcflow*( ) function may fail if:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE ALSO

tcsendbreak( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

tcflush — flush non-transmitted output data, non-read input data, or both

SYNOPSIS

```
#include <termios.h>

int tcflush(int fildes, int queue_selector);
```

DESCRIPTION

Upon successful completion, *tcflush*( ) shall discard data written to the object referred to by *fildes* (an open file descriptor associated with a terminal) but not transmitted, or data received but not read, depending on the value of *queue_selector*:

- If *queue_selector* is TCIFLUSH, it shall flush data received but not read.
- If *queue_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- If *queue_selector* is TCIOFLUSH, it shall flush both data received but not read and data written but not transmitted.

Attempts to use *tcflush*( ) from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcflush*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] The *queue_selector* argument is not a supported value.

[ENOTTY] The file associated with *fildes* is not a terminal.

The *tcflush*( ) function may fail if:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcdrain( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**, **<unistd.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE sections, references to *tcflow*( ) are replaced with *tcflush*( ).

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the final paragraph is no longer conditional on _POSIX_JOB_CONTROL. This is a FIPS requirement.
- The [EIO] error is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

46302 **NAME**

46303 tcgetattr — get the parameters associated with the terminal

46304 **SYNOPSIS**

46305 `#include <termios.h>`

46306 `int tcgetattr(int fildes, struct termios *termios_p);`

46307 **DESCRIPTION**

46308 The *tcgetattr*( ) function shall get the parameters associated with the terminal referred to by *fildes*
46309 and store them in the **termios** structure referenced by *termios_p*. The *fildes* argument is an open
46310 file descriptor associated with a terminal.

46311 The *termios_p* argument is a pointer to a **termios** structure.

46312 The *tcgetattr*( ) operation is allowed from any process.

46313 If the terminal device supports different input and output baud rates, the baud rates stored in
46314 the **termios** structure returned by *tcgetattr*( ) shall reflect the actual baud rates, even if they are
46315 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be
46316 the actual baud rate. If the terminal device does not support split baud rates, the input baud rate
46317 stored in the **termios** structure shall be the output rate (as one of the symbolic values).

46318 **RETURN VALUE**

46319 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
46320 indicate the error.

46321 **ERRORS**

46322 The *tcgetattr*( ) function shall fail if:

46323 [EBADF] The *fildes* argument is not a valid file descriptor.

46324 [ENOTTY] The file associated with *fildes* is not a terminal.

46325 **EXAMPLES**

46326 None.

46327 **APPLICATION USAGE**

46328 None.

46329 **RATIONALE**

46330 Care must be taken when changing the terminal attributes. Applications should always do a
46331 *tcgetattr*( ), save the **termios** structure values returned, and then do a *tcsetattr*( ), changing only
46332 the necessary fields. The application should use the values saved from the *tcgetattr*( ) to reset the
46333 terminal state whenever it is done with the terminal. This is necessary because terminal
46334 attributes apply to the underlying port and not to each individual open instance; that is, all
46335 processes that have used the terminal see the latest attribute changes.

46336 A program that uses these functions should be written to catch all signals and take other
46337 appropriate actions to ensure that when the program terminates, whether planned or not, the
46338 terminal device's state is restored to its original state.

46339 Existing practice dealing with error returns when only part of a request can be honored is based
46340 on calls to the *ioctl*( ) function. In historical BSD and System V implementations, the
46341 corresponding *ioctl*( ) returns zero if the requested actions were semantically correct, even if
46342 some of the requested changes could not be made. Many existing applications assume this
46343 behavior and would no longer work correctly if the return value were changed from zero to –1
46344 in this case.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note that either specification has a problem. When zero is returned, it implies everything succeeded even if some of the changes were not made. When –1 is returned, it implies everything failed even though some of the changes were made.

Applications that need all of the requested changes made to work properly should follow *tcsetattr*( ) with a call to *tcgetattr*( ) and compare the appropriate field values.

FUTURE DIRECTIONS

None.

SEE ALSO

tcsetattr( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate. Previously, the number zero was also allowed but was obsolescent.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

46360 **NAME**

46361 tcgetpgrp — get the foreground process group ID

46362 **SYNOPSIS**

46363 `#include <unistd.h>`

46364 `pid_t tcgetpgrp(int fildes);`

46365 **DESCRIPTION**

46366 The *tcgetpgrp*( ) function shall return the value of the process group ID of the foreground process
46367 group associated with the terminal.

46368 If there is no foreground process group, *tcgetpgrp*( ) shall return a value greater than 1 that does
46369 not match the process group ID of any existing process group.

46370 The *tcgetpgrp*( ) function is allowed from a process that is a member of a background process
46371 group; however, the information may be subsequently changed by a process that is a member of
46372 a foreground process group.

46373 **RETURN VALUE**

46374 Upon successful completion, *tcgetpgrp*( ) shall return the value of the process group ID of the
46375 foreground process associated with the terminal. Otherwise, –1 shall be returned and *errno* set to
46376 indicate the error.

46377 **ERRORS**

46378 The *tcgetpgrp*( ) function shall fail if:

46379 [EBADF] The *fildes* argument is not a valid file descriptor.

46380 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
46381 controlling terminal.

46382 **EXAMPLES**

46383 None.

46384 **APPLICATION USAGE**

46385 None.

46386 **RATIONALE**

46387 None.

46388 **FUTURE DIRECTIONS**

46389 None.

46390 **SEE ALSO**

46391 *setsid*( ), *setpgid*( ), *tcsetpgrp*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
46392 **<sys/types.h>**, **<unistd.h>**

46393 **CHANGE HISTORY**

46394 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46395 **Issue 6**

46396 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

46397 The following new requirements on POSIX implementations derive from alignment with the
46398 Single UNIX Specification:

46399 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
46400 required for conforming implementations of previous POSIX specifications, it was not
46401 required for UNIX applications.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

46402 • In the DESCRIPTION, text previously conditional on support for _POSIX_JOB_CONTROL is
46403 now mandatory. This is a FIPS requirement.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tcgetsid — get the process group ID for the session leader for the controlling terminal

SYNOPSIS

XSI
```
#include <termios.h>

pid_t tcgetsid(int fildes);
```

DESCRIPTION

The *tcgetsid*( ) function shall obtain the process group ID of the session for which the terminal specified by *fildes* is the controlling terminal.

RETURN VALUE

Upon successful completion, *tcgetsid*( ) shall return the process group ID associated with the terminal. Otherwise, a value of (**pid_t**)–1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcgetsid*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[ENOTTY] The calling process does not have a controlling terminal, or the file is not the controlling terminal.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<termios.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The [EACCES] error has been removed from the list of mandatory errors, and the description of [ENOTTY] has been reworded.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tcsendbreak — send a break for a specific duration

SYNOPSIS

```
#include <termios.h>

int tcsendbreak(int fildes, int duration);
```

DESCRIPTION

If the terminal is using asynchronous serial data transmission, *tcsendbreak*( ) shall cause transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period of time.

The *fildes* argument is an open file descriptor associated with a terminal.

If the terminal is not using asynchronous serial data transmission, it is implementation-defined whether *tcsendbreak*( ) sends data to generate a break condition or returns without taking any action.

Attempts to use *tcsendbreak*( ) from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsendbreak*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[ENOTTY] The file associated with *fildes* is not a terminal.

The *tcsendbreak*( ) function may fail if:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**, **<unistd.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

46477 **CHANGE HISTORY**

46478 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46479 **Issue 6**

46480 The following new requirements on POSIX implementations derive from alignment with the
46481 Single UNIX Specification:

46482 • In the DESCRIPTION, text previously conditional on _POSIX_JOB_CONTROL is now
46483 mandated. This is a FIPS requirement.

46484 • The [EIO] error is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

46485 **NAME**

46486 tcsetattr — set the parameters associated with the terminal

46487 **SYNOPSIS**

46488
```
#include <termios.h>
```

46489
```
int tcsetattr(int fildes, int optional_actions,
```
46490
```
    const struct termios *termios_p);
```

46491 **DESCRIPTION**

46492 The *tcsetattr*( ) function shall set the parameters associated with the terminal referred to by the
46493 open file descriptor *fildes* (an open file descriptor associated with a terminal) from the **termios**
46494 structure referenced by *termios_p* as follows:

46495 • If *optional_actions* is TCSANOW, the change shall occur immediately.

46496 • If *optional_actions* is TCSADRAIN, the change shall occur after all output written to *fildes* is
46497 transmitted. This function should be used when changing parameters that affect output.

46498 • If *optional_actions* is TCSAFLUSH, the change shall occur after all output written to *fildes* is
46499 transmitted, and all input so far received but not read shall be discarded before the change is
46500 made.

46501 If the output baud rate stored in the **termios** structure pointed to by *termios_p* is the zero baud
46502 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the
46503 line.

46504 If the input baud rate stored in the **termios** structure pointed to by *termios_p* is 0, the input baud
46505 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

46506 The *tcsetattr*( ) function shall return successfully if it was able to perform any of the requested
46507 actions, even if some of the requested actions could not be performed. It shall set all the
46508 attributes that the implementation supports as requested and leave all the attributes not
46509 supported by the implementation unchanged. If no part of the request can be honored, it shall
46510 return −1 and set *errno* to [EINVAL]. If the input and output baud rates differ and are a
46511 combination that is not supported, neither baud rate shall be changed. A subsequent call to
46512 *tcgetattr*( ) shall return the actual state of the terminal device (reflecting both the changes made
46513 and not made in the previous *tcsetattr*( ) call). The *tcsetattr*( ) function shall not change the values
46514 found in the **termios** structure under any circumstances.

46515 The effect of *tcsetattr*( ) is undefined if the value of the **termios** structure pointed to by *termios_p*
46516 was not derived from the result of a call to *tcgetattr*( ) on *fildes*; an application should modify
46517 only fields and flags defined by this volume of IEEE Std 1003.1-2001 between the call to
46518 *tcgetattr*( ) and *tcsetattr*( ), leaving all other fields and flags unmodified.

46519 No actions defined by this volume of IEEE Std 1003.1-2001, other than a call to *tcsetattr*( ) or a
46520 close of the last file descriptor in the system associated with this terminal device, shall cause any
46521 of the terminal attributes defined by this volume of IEEE Std 1003.1-2001 to change.

46522 If *tcsetattr*( ) is called from a process which is a member of a background process group on a
46523 *fildes* associated with its controlling terminal:

46524 • If the calling process is blocking or ignoring SIGTTOU signals, the operation completes
46525 normally and no signal is sent.

46526 • Otherwise, a SIGTTOU signal shall be sent to the process group.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsetattr*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINTR] A signal interrupted *tcsetattr*( ).

[EINVAL] The *optional_actions* argument is not a supported value, or an attempt was made to change an attribute represented in the **termios** structure to an unsupported value.

[ENOTTY] The file associated with *fildes* is not a terminal.

The *tcsetattr*( ) function may fail if:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

EXAMPLES

None.

APPLICATION USAGE

If trying to change baud rates, applications should call *tcsetattr*( ) then call *tcgetattr*( ) in order to determine what baud rates were actually selected.

RATIONALE

The *tcsetattr*( ) function can be interrupted in the following situations:

- It is interrupted while waiting for output to drain.
- It is called from a process in a background process group and SIGTTOU is caught.

See also the RATIONALE section in *tcgetattr*( ).

FUTURE DIRECTIONS

Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be supported in a future version of this volume of IEEE Std 1003.1-2001.

SEE ALSO

cfgetispeed( ), *tcgetattr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text previously conditional on _POSIX_JOB_CONTROL is now mandated. This is a FIPS requirement.
- The [EIO] error is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

46565 In the DESCRIPTION, the text describing use of *tcsetattr*( ) from a process which is a member of
46566 a background process group is clarified.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

tcsetpgrp — set the foreground process group ID

SYNOPSIS

```
#include <unistd.h>

int tcsetpgrp(int fildes, pid_t pgid_id);
```

DESCRIPTION

If the process has a controlling terminal, *tcsetpgrp*( ) shall set the foreground process group ID associated with the terminal to *pgid_id*. The application shall ensure that the file associated with *fildes* is the controlling terminal of the calling process and the controlling terminal is currently associated with the session of the calling process. The application shall ensure that the value of *pgid_id* matches a process group ID of a process in the same session as the calling process.

Attempts to use *tcsetpgrp*( ) from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsetpgrp*( ) function shall fail if:

[EBADF] The *fildes* argument is not a valid file descriptor.

[EINVAL] This implementation does not support the value in the *pgid_id* argument.

[ENOTTY] The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.

[EPERM] The value of *pgid_id* is a value supported by the implementation, but does not match the process group ID of a process in the same session as the calling process.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcgetpgrp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the DESCRIPTION and ERRORS sections, text previously conditional on _POSIX_JOB_CONTROL is now mandated. This is a FIPS requirement.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The Open Group Corrigendum U047/4 is applied.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

tdelete, tfind, tsearch, twalk — manage a binary search tree

SYNOPSIS

XSI
```
#include <search.h>

void *tdelete(const void *restrict key, void **restrict rootp,
    int(*compar)(const void *, const void *));
void *tfind(const void *key, void *const *rootp,
    int(*compar)(const void *, const void *));
void *tsearch(const void *key, void **rootp,
    int (*compar)(const void *, const void *));
void twalk(const void *root,
    void (*action)(const void *, VISIT, int));
```

DESCRIPTION

The *tdelete*(), *tfind*(), *tsearch*(), and *twalk*() functions manipulate binary search trees. Comparisons are made with a user-supplied routine, the address of which is passed as the *compar* argument. This routine is called with two arguments, which are the pointers to the elements being compared. The application shall ensure that the user-supplied routine returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to, or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The *tsearch*() function shall build and access the tree. The *key* argument is a pointer to an element to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key* shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a pointer to this node returned. Only pointers are copied, so the application shall ensure that the calling routine stores the data. The *rootp* argument points to a variable that points to the root node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree; in this case, the variable shall be set to point to the node which shall be at the root of the new tree.

Like *tsearch*(), *tfind*() shall search for a node in the tree, returning a pointer to it if found. However, if it is not found, *tfind*() shall return a null pointer. The arguments for *tfind*() are the same as for *tsearch*().

The *tdelete*() function shall delete a node from a binary search tree. The arguments are the same as for *tsearch*(). The variable pointed to by *rootp* shall be changed if the deleted node was the root of the tree. The *tdelete*() function shall return a pointer to the parent of the deleted node, or a null pointer if the node is not found.

The *twalk*() function shall traverse a binary search tree. The *root* argument is a pointer to the root node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) The argument *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument shall be the address of the node being visited. The structure pointed to by this argument is unspecified and shall not be modified by the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-element to access the element stored in the node. The second argument shall be a value from an enumeration data type:

```
typedef enum { preorder, postorder, endorder, leaf } VISIT;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(defined in **<search.h>**), depending on whether this is the first, second, or third time that the node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

If the calling function alters the pointer to the root, the result is undefined.

RETURN VALUE

If the node is found, both *tsearch*( ) and *tfind*( ) shall return a pointer to it. If not, *tfind*( ) shall return a null pointer, and *tsearch*( ) shall return a pointer to the inserted item.

A null pointer shall be returned by *tsearch*( ) if there is not enough space available to create a new node.

A null pointer shall be returned by *tdelete*( ), *tfind*( ), and *tsearch*( ) if *rootp* is a null pointer on entry.

The *tdelete*( ) function shall return a pointer to the parent of the deleted node, or a null pointer if the node is not found.

The *twalk*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <string.h>
#include <stdio.h>

#define STRSZ    10000
#define NODSZ    500

struct node {      /* Pointers to these are stored in the tree. */
    char    *string;
    int     length;
};

char   string_space[STRSZ];  /* Space to store strings. */
struct node nodes[NODSZ];    /* Nodes to store. */
void  *root = NULL;          /* This points to the root. */

int main(int argc, char *argv[])
{
    char   *strptr = string_space;
    struct node    *nodeptr = nodes;
    void   print_node(const void *, VISIT, int);
    int    i = 0, node_compare(const void *, const void *);

    while (gets(strptr) != NULL && i++ < NODSZ)  {
        /* Set node. */
        nodeptr->string = strptr;
        nodeptr->length = strlen(strptr);
        /* Put node into the tree. */
        (void) tsearch((void *)nodeptr, (void **)&root,
            node_compare);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
46710               /* Adjust pointers, so we do not overwrite tree. */
46711               strptr += nodeptr->length + 1;
46712               nodeptr++;
46713           }
46714           twalk(root, print_node);
46715           return 0;
46716       }
46717       /*
46718        *  This routine compares two nodes, based on an
46719        *  alphabetical ordering of the string field.
46720        */
46721       int
46722       node_compare(const void *node1, const void *node2)
46723       {
46724           return strcmp(((const struct node *) node1)->string,
46725               ((const struct node *) node2)->string);
46726       }
46727       /*
46728        *  This routine prints out a node, the second time
46729        *  twalk encounters it or if it is a leaf.
46730        */
46731       void
46732       print_node(const void *ptr, VISIT order, int level)
46733       {
46734           const struct node *p = *(const struct node **) ptr;
46735           if (order == postorder || order == leaf)  {
46736               (void) printf("string = %s,  length = %d\n",
46737                   p->string, p->length);
46738           }
46739       }
```

46740 **APPLICATION USAGE**

46741 The *root* argument to *twalk*( ) is one level of indirection less than the *rootp* arguments to *tdelete*( )
46742 and *tsearch*( ).

46743 There are two nomenclatures used to refer to the order in which tree nodes are visited. The
46744 *tsearch*( ) function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node
46745 before any of its children, after its left child and before its right, and after both its children. The
46746 alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which
46747 could result in some confusion over the meaning of **postorder**.

46748 **RATIONALE**

46749 None.

46750 **FUTURE DIRECTIONS**

46751 None.

46752 **SEE ALSO**

46753 *hcreate*( ), *lsearch*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<search.h>**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The **restrict** keyword is added to the *tdelete*() prototype for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

telldir — current location of a named directory stream

SYNOPSIS

XSI
```
#include <dirent.h>

long telldir(DIR *dirp);
```

DESCRIPTION

The *telldir*( ) function shall obtain the current location associated with the directory stream specified by *dirp*.

If the most recent operation on the directory stream was a *seekdir*( ), the directory position returned from the *telldir*( ) shall be the same as that supplied as a *loc* argument for *seekdir*( ).

RETURN VALUE

Upon successful completion, *telldir*( ) shall return the current location of the specified directory stream.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

opendir( ), *readdir*( ), *seekdir*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**

CHANGE HISTORY

First released in Issue 2.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tempnam — create a name for a temporary file

SYNOPSIS

XSI
```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx);
```

DESCRIPTION

The *tempnam*( ) function shall generate a pathname that may be used for a temporary file.

The *tempnam*( ) function allows the user to control the choice of a directory. The *dir* argument points to the name of the directory in which the file is to be created. If *dir* is a null pointer or points to a string which is not a name for an appropriate directory, the path prefix defined as P_tmpdir in the **<stdio.h>** header shall be used. If that directory is not accessible, an implementation-defined directory may be used.

Many applications prefer their temporary files to have certain initial letter sequences in their names. The *pfx* argument should be used for this. This argument may be a null pointer or point to a string of up to five bytes to be used as the beginning of the filename.

Some implementations of *tempnam*( ) may use *tmpnam*( ) internally. On such implementations, if called more than {TMP_MAX} times in a single process, the behavior is implementation-defined.

RETURN VALUE

Upon successful completion, *tempnam*( ) shall allocate space for a string, put the generated pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a subsequent call to *free*( ). Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *tempnam*( ) function shall fail if:

[ENOMEM] Insufficient storage space is available.

EXAMPLES

Generating a Pathname

The following example generates a pathname for a temporary file in directory **/tmp**, with the prefix *file*. After the filename has been created, the call to *free*( ) deallocates the space used to store the filename.

```
#include <stdio.h>
#include <stdlib.h>
...
char *directory = "/tmp";
char *fileprefix = "file";
char *file;

file = tempnam(directory, fileprefix);
free(file);
```

APPLICATION USAGE

This function only creates pathnames. It is the application's responsibility to create and remove the files. Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find *tmpfile*( ) more useful.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

46834 **RATIONALE**

46835 None.

46836 **FUTURE DIRECTIONS**

46837 None.

46838 **SEE ALSO**

46839 *fopen*( ), *free*( ), *open*( ), *tmpfile*( ), *tmpnam*( ), *unlink*( ), the Base Definitions volume of
46840 IEEE Std 1003.1-2001, **<stdio.h>**

46841 **CHANGE HISTORY**

46842 First released in Issue 1. Derived from Issue 1 of the SVID.

46843 **Issue 5**

46844 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
46845 previous issues.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tfind — search binary search tree

SYNOPSIS

XSI

```
#include <search.h>

void *tfind(const void *key, void *const *rootp,
    int (*compar)(const void *, const void *));
```

DESCRIPTION

Refer to *tdelete*( ).

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tgamma, tgammaf, tgammal — compute gamma( ) function

SYNOPSIS

```
#include <math.h>

double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall compute the *gamma*( ) function of *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return *Gamma*(*x*).

If *x* is a negative integer, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If the correct value would cause overflow, a range error shall occur and *tgamma*( ), *tgammaf*( ), and *tgammal*( ) shall return ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

MX If *x* is NaN, a NaN shall be returned.

If *x* is +Inf, *x* shall be returned.

If *x* is ±0, a pole error shall occur, and *tgamma*( ), *tgammaf*( ), and *tgammal*( ) shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively.

If *x* is –Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

ERRORS

These functions shall fail if:

MX **Domain Error** The value of *x* is a negative integer, or *x* is –Inf.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

MX **Pole Error** The value of *x* is zero.

If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

46896	Range Error	The value overflows.
46897		If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
46898		then *errno* shall be set to [ERANGE]. If the integer expression
46899		(math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
46900		floating-point exception shall be raised.

46901 **EXAMPLES**

46902 None.

46903 **APPLICATION USAGE**

46904 For IEEE Std 754-1985 **double**, overflow happens when $0 < x < 1/\text{DBL_MAX}$, and $171.7 < x$.

46905 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
46906 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46907 **RATIONALE**

46908 This function is named *tgamma*( ) in order to avoid conflicts with the historical *gamma*( ) and
46909 *lgamma*( ) functions.

46910 **FUTURE DIRECTIONS**

46911 It is possible that the error response for a negative integer argument may be changed to a pole
46912 error and a return value of ±Inf.

46913 **SEE ALSO**

46914 *feclearexcept*( ), *fetestexcept*( ), *lgamma*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
46915 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

46916 **CHANGE HISTORY**

46917 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

46918 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 65 is applied, correcting the third
46919 paragraph in the RETURN VALUE section.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

time — get time

SYNOPSIS

```
#include <time.h>

time_t time(time_t *tloc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

CX The *time*( ) function shall return the value of time in seconds since the Epoch.

The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer, no value is stored.

RETURN VALUE

Upon successful completion, *time*( ) shall return the value of time. Otherwise, (**time_t**)–1 shall be returned.

ERRORS

No errors are defined.

EXAMPLES

Getting the Current Time

The following example uses the *time*( ) function to calculate the time elapsed, in seconds, since the Epoch, *localtime*( ) to convert that value to a broken-down time, and *asctime*( ) to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
time_t result;

    result = time(NULL);
    printf("%s%ju secs since the Epoch\n",
        asctime(localtime(&result)),
            (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Timing an Event

The following example gets the current time, prints it out in the user's format, and prints the number of minutes to an event being timed.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
minutes_to_event = ...;
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",
    minutes_to_event);
...
```

APPLICATION USAGE

None.

RATIONALE

The *time*( ) function returns a value in seconds (type **time_t**) while *times*( ) returns a set of values in clock ticks (type **clock_t**). Some historical implementations, such as 4.3 BSD, have mechanisms capable of returning more precise times (see below). A generalized timing scheme to unify these various timing mechanisms has been proposed but not adopted.

Implementations in which **time_t** is a 32-bit signed integer (many historical implementations) fail in the year 2038. IEEE Std 1003.1-2001 does not address this problem. However, the use of the **time_t** type is mandated in order to ease the eventual fix.

The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C standard.

Many historical implementations (including Version 7) and the 1984 /usr/group standard use **long** instead of **time_t**. This volume of IEEE Std 1003.1-2001 uses the latter type in order to agree with the ISO C standard.

4.3 BSD includes *time*( ) only as an alternate function to the more flexible *gettimeofday*( ) function.

FUTURE DIRECTIONS

In a future version of this volume of IEEE Std 1003.1-2001, **time_t** is likely to be required to be capable of representing times far in the future. Whether this will be mandated as a 64-bit type or a requirement that a specific date in the future be representable (for example, 10000 AD) is not yet determined. Systems purchased after the approval of this volume of IEEE Std 1003.1-2001 should be evaluated to determine whether their lifetime will extend past 2038.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *gettimeofday*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *utime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

46999 **Issue 6**

47000 Extensions beyond the ISO C standard are marked.

47001 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

47002 **NAME**

47003 timer_create — create a per-process timer (**REALTIME**)

47004 **SYNOPSIS**

47005 TMR
```
#include <signal.h>
#include <time.h>

int timer_create(clockid_t clockid, struct sigevent *restrict evp,
    timer_t *restrict timerid);
```
47009

47010 **DESCRIPTION**

47011 The *timer_create*( ) function shall create a per-process timer using the specified clock, *clock_id*, as
47012 the timing base. The *timer_create*( ) function shall return, in the location referenced by *timerid*, a
47013 timer ID of type **timer_t** used to identify the timer in timer requests. This timer ID shall be
47014 unique within the calling process until the timer is deleted. The particular clock, *clock_id*, is
47015 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return
47016 from *timer_create*( ).

47017 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the
47018 application, defines the asynchronous notification to occur as specified in Section 2.4.1 (on page
47019 28) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument
47020 pointed to a **sigevent** structure with the *sigev_notify* member having the value SIGEV_SIGNAL,
47021 the *sigev_signo* having a default signal number, and the *sigev_value* member having the value of
47022 the timer ID.

47023 Each implementation shall define a set of clocks that can be used as timing bases for per-process
47024 MON timers. All implementations shall support a *clock_id* of CLOCK_REALTIME. If the Monotonic
47025 Clock option is supported, implementations shall support a *clock_id* of CLOCK_MONOTONIC.

47026 Per-process timers shall not be inherited by a child process across a *fork*( ) and shall be disarmed
47027 and deleted by an *exec*.

47028 CPT If _POSIX_CPUTIME is defined, implementations shall support *clock_id* values representing the
47029 CPU-time clock of the calling process.

47030 TCT If _POSIX_THREAD_CPUTIME is defined, implementations shall support *clock_id* values
47031 representing the CPU-time clock of the calling thread.

47032 CPT|TCT It is implementation-defined whether a *timer_create*( ) function will succeed if the value defined
47033 by *clock_id* corresponds to the CPU-time clock of a process or thread different from the process
47034 or thread invoking the function.

47035 **RETURN VALUE**

47036 If the call succeeds, *timer_create*( ) shall return zero and update the location referenced by *timerid*
47037 to a **timer_t**, which can be passed to the per-process timer calls. If an error occurs, the function
47038 shall return a value of –1 and set *errno* to indicate the error. The value of *timerid* is undefined if
47039 an error occurs.

47040 **ERRORS**

47041 The *timer_create*( ) function shall fail if:

47042 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

47043 [EAGAIN] The calling process has already created all of the timers it is allowed by this
47044 implementation.

47045 [EINVAL] The specified clock ID is not defined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CPT|TCT [ENOTSUP] The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by *clock_id* and associated with a process or thread different from the process or thread invoking *timer_create*( ).

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Periodic Timer Overrun and Resource Allocation

The specified timer facilities may deliver realtime signals (that is, queued signals) on implementations that support this option. Since realtime applications cannot afford to lose notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it must be possible to ensure that sufficient resources exist to deliver the signal when the event occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a request and a subsequent signal generation. If the request cannot allocate the signal delivery resources, it can fail the call with an [EAGAIN] error.

Periodic timers are a special case. A single request can generate an unspecified number of signals. This is not a problem if the requesting process can service the signals as fast as they are generated, thus making the signal delivery resources available for delivery of subsequent periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic timer signals may ‘‘overrun’’; that is, subsequent periodic timer expirations may occur before the currently pending signal has been delivered.

Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a pending signal are generated, it is implementation-defined whether a signal is delivered for each occurrence. This is not adequate for some realtime applications. So a mechanism is required to allow applications to detect how many timer expirations were delayed without requiring an indefinite amount of system resources to store the delayed expirations.

The specified facilities provide for an overrun count. The overrun count is defined as the number of extra timer expirations that occurred between the time a timer expiration signal is generated and the time the signal is delivered. The signal-catching function, if it is concerned with overruns, can retrieve this count on entry. With this method, a periodic timer only needs one ‘‘signal queuing resource’’ that can be allocated at the time of the *timer_create*( ) function call.

A function is defined to retrieve the overrun count so that an application need not allocate static storage to contain the count, and an implementation need not update this storage asynchronously on timer expirations. But, for some high-frequency periodic applications, the overhead of an additional system call on each timer expiration may be prohibitive. The functions, as defined, permit an implementation to maintain the overrun count in user space, associated with the *timerid*. The *timer_getoverrun*( ) function can then be implemented as a macro that uses the *timerid* argument (which may just be a pointer to a user space structure containing the counter) to locate the overrun count with no system call overhead. Other implementations, less concerned with this class of applications, can avoid the asynchronous update of user space by maintaining the count in a system structure at the cost of the extra system call to obtain it.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

47088 **Timer Expiration Signal Parameters**

47089 The Realtime Signals Extension option supports an application-specific datum that is delivered
47090 to the extended signal handler. This value is explicitly specified by the application, along with
47091 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined
47092 value can be either an integer constant or a pointer. This explicit specification of the value, as
47093 opposed to always sending the timer ID, was selected based on existing practice.

47094 It is common practice for realtime applications (on non-POSIX systems or realtime extended
47095 POSIX systems) to use the parameters of event handlers as the case label of a switch statement
47096 or as a pointer to an application-defined data structure. Since *timer_id*s are dynamically allocated
47097 by the *timer_create*( ) function, they can be used for neither of these functions without additional
47098 application overhead in the signal handler; for example, to search an array of saved timer IDs to
47099 associate the ID with a constant or application data structure.

47100 **FUTURE DIRECTIONS**

47101 None.

47102 **SEE ALSO**

47103 *clock_getres*( ), *timer_delete*( ), *timer_getoverrun*( ), the Base Definitions volume of
47104 IEEE Std 1003.1-2001, **<time.h>**

47105 **CHANGE HISTORY**

47106 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47107 **Issue 6**

47108 The *timer_create*( ) function is marked as part of the Timers option.

47109 The [ENOSYS] error condition has been removed as stubs need not be provided if an
47110 implementation does not support the Timers option.

47111 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

47112 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the
47113 requirement for the CLOCK_MONOTONIC clock under the Monotonic Clock option.

47114 The **restrict** keyword is added to the *timer_create*( ) prototype for alignment with the
47115 ISO / IEC 9899: 1999 standard.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

47116 **NAME**

47117 timer_delete — delete a per-process timer (**REALTIME**)

47118 **SYNOPSIS**

47119 TMR
```
#include <time.h>
```

47120
```
int timer_delete(timer_t timerid);
```

47121

47122 **DESCRIPTION**

47123 The *timer_delete*( ) function deletes the specified timer, *timerid*, previously created by the
47124 *timer_create*( ) function. If the timer is armed when *timer_delete*( ) is called, the behavior shall be
47125 as if the timer is automatically disarmed before removal. The disposition of pending signals for
47126 the deleted timer is unspecified.

47127 **RETURN VALUE**

47128 If successful, the *timer_delete*( ) function shall return a value of zero. Otherwise, the function shall
47129 return a value of –1 and set *errno* to indicate the error.

47130 **ERRORS**

47131 The *timer_delete*( ) function shall fail if:

47132 [EINVAL] The timer ID specified by *timerid* is not a valid timer ID.

47133 **EXAMPLES**

47134 None.

47135 **APPLICATION USAGE**

47136 None.

47137 **RATIONALE**

47138 None.

47139 **FUTURE DIRECTIONS**

47140 None.

47141 **SEE ALSO**

47142 *timer_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

47143 **CHANGE HISTORY**

47144 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47145 **Issue 6**

47146 The *timer_delete*( ) function is marked as part of the Timers option.

47147 The [ENOSYS] error condition has been removed as stubs need not be provided if an
47148 implementation does not support the Timers option.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

timer_getoverrun, timer_gettime, timer_settime — per-process timers (**REALTIME**)

SYNOPSIS

TMR
```
#include <time.h>

int timer_getoverrun(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_settime(timer_t timerid, int flags,
    const struct itimerspec *restrict value,
    struct itimerspec *restrict ovalue);
```

DESCRIPTION

The *timer_gettime*( ) function shall store the amount of time until the specified timer, *timerid*, expires and the reload value of the timer into the space pointed to by the *value* argument. The *it_value* member of this structure shall contain the amount of time before the timer expires, or zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if the timer was armed with absolute time. The *it_interval* member of *value* shall contain the reload value last set by *timer_settime*( ).

The *timer_settime*( ) function shall set the time until the next expiration of the timer specified by *timerid* from the *it_value* member of the *value* argument and arm the timer if the *it_value* member of *value* is non-zero. If the specified timer was already armed when *timer_settime*( ) is called, this call shall reset the time until next expiration to the *value* specified. If the *it_value* member of *value* is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending expiration notifications is unspecified.

If the flag TIMER_ABSTIME is not set in the argument *flags*, *timer_settime*( ) shall behave as if the time until next expiration is set to be equal to the interval specified by the *it_value* member of *value*. That is, the timer shall expire in *it_value* nanoseconds from when the call is made. If the flag TIMER_ABSTIME is set in the argument *flags*, *timer_settime*( ) shall behave as if the time until next expiration is set to be equal to the difference between the absolute time specified by the *it_value* member of *value* and the current value of the clock associated with *timerid*. That is, the timer shall expire when the clock reaches the value specified by the *it_value* member of *value*. If the specified time has already passed, the function shall succeed and the expiration notification shall be made.

The reload value of the timer shall be set to the value specified by the *it_interval* member of *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is specified.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization error shall not cause the timer to expire earlier than the rounded time value.

If the argument *ovalue* is not NULL, the *timer_settime*( ) function shall store, in the location referenced by *ovalue*, a value representing the previous amount of time before the timer would have expired, or zero if the timer was disarmed, together with the previous timer reload value. Timers shall not expire before their scheduled time.

Only a single signal shall be queued to the process for a given timer at any point in time. When a timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun shall occur. RTS When a timer expiration signal is delivered to or accepted by a process, if the implementation supports the Realtime Signals Extension, the *timer_getoverrun*( ) function shall return the timer expiration overrun count for the specified timer. The overrun count returned contains the number of extra timer expirations that occurred between the time the signal was

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

generated (queued) and when it was delivered or accepted, up to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the overrun count shall be set to {DELAYTIMER_MAX}. The value returned by *timer_getoverrun*( ) shall apply to the most recent expiration signal delivery or acceptance for the timer. If no expiration signal has been delivered for the timer, or if the Realtime Signals Extension is not supported, the return value of *timer_getoverrun*( ) is unspecified.

RETURN VALUE

If the *timer_getoverrun*( ) function succeeds, it shall return the timer expiration overrun count as explained above.

If the *timer_gettime*( ) or *timer_settime*( ) functions succeed, a value of 0 shall be returned.

If an error occurs for any of these functions, the value –1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *timer_getoverrun*( ), *timer_gettime*( ), and *timer_settime*( ) functions shall fail if:

[EINVAL] The *timerid* argument does not correspond to an ID returned by *timer_create*( ) but not yet deleted by *timer_delete*( ).

The *timer_settime*( ) function shall fail if:

[EINVAL] A *value* structure specified a nanosecond value less than zero or greater than or equal to 1 000 million, and the *it_value* member of that structure did not specify zero seconds and nanoseconds.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The inverse of this tick rate is the clock resolution, also called the clock granularity, which in either case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for these common rates. The granularity of practical clocks implies that if one reads a given clock twice in rapid succession, one may get the same time value twice; and that timers must wait for the next clock tick after the theoretical expiration time, to ensure that a timer never returns too soon. Note also that the granularity of the clock may be significantly coarser than the resolution of the data format used to set and get time and interval values. Also note that some implementations may choose to adjust time and/or interval values to exactly match the ticks of the underlying clock.

This volume of IEEE Std 1003.1-2001 defines functions that allow an application to determine the implementation-supported resolution for the clocks and requires an implementation to document the resolution supported for timers and *nanosleep*( ) if they differ from the supported clock resolution. This is more of a procurement issue than a runtime application issue.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

47239 **SEE ALSO**

47240 *clock_getres*( ), *timer_create*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

47241 **CHANGE HISTORY**

47242 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47243 **Issue 6**

47244 The *timer_getoverrun*( ), *timer_gettime*( ), and *timer_settime*( ) functions are marked as part of the
47245 Timers option.

47246 The [ENOSYS] error condition has been removed as stubs need not be provided if an
47247 implementation does not support the Timers option.

47248 The [EINVAL] error condition is updated to include the following: ''and the *it_value* member of
47249 that structure did not specify zero seconds and nanoseconds.'' This change is for IEEE PASC
47250 Interpretation 1003.1 #89.

47251 The DESCRIPTION for *timer_getoverrun*( ) is updated to clarify that ''If no expiration signal has
47252 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return
47253 value of *timer_getoverrun*( ) is unspecified''.

47254 The **restrict** keyword is added to the *timer_settime*( ) prototype for alignment with the
47255 ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

times — get process and waited-for child process times

SYNOPSIS

```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

DESCRIPTION

The *times*( ) function shall fill the **tms** structure pointed to by *buffer* with time-accounting information. The **tms** structure is defined in **<sys/times.h>**.

All times are measured in terms of the number of clock ticks used.

The times of a terminated child process shall be included in the *tms_cutime* and *tms_cstime* elements of the parent when *wait*( ) or *waitpid*( ) returns the process ID of this terminated child. If a child process has not waited for its children, their times shall not be included in its times.

- The *tms_utime* structure member is the CPU time charged for the execution of user instructions of the calling process.
- The *tms_stime* structure member is the CPU time charged for execution by the system on behalf of the calling process.
- The *tms_cutime* structure member is the sum of the *tms_utime* and *tms_cutime* times of the child processes.
- The *tms_cstime* structure member is the sum of the *tms_stime* and *tms_cstime* times of the child processes.

RETURN VALUE

Upon successful completion, *times*( ) shall return the elapsed real time, in clock ticks, since an arbitrary point in the past (for example, system start-up time). This point does not change from one invocation of *times*( ) within the process to another. The return value may overflow the possible range of type **clock_t**. If *times*( ) fails, (**clock_t**)–1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

Timing a Database Lookup

The following example defines two functions, *start_clock*( ) and *end_clock*( ), that are used to time a lookup. It also defines variables of type **clock_t** and **tms** to measure the duration of transactions. The *start_clock*( ) function saves the beginning times given by the *times*( ) function. The *end_clock*( ) function gets the ending times and prints the difference between the two times.

```
#include <sys/times.h>
#include <stdio.h>
...
void start_clock(void);
void end_clock(char *msg);
...
static clock_t st_time;
static clock_t en_time;
static struct tms st_cpu;
static struct tms en_cpu;
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
47300       ...
47301       void
47302       start_clock()
47303       {
47304           st_time = times(&st_cpu);
47305       }

47306       /* This example assumes that the result of each subtraction
47307          is within the range of values that can be represented in
47308          an integer type. */
47309       void
47310       end_clock(char *msg)
47311       {
47312           en_time = times(&en_cpu);

47313           fputs(msg,stdout);
47314           printf("Real Time: %jd, User Time %jd, System Time %jd\n",
47315               (intmax_t)(en_time - st_time),
47316               (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
47317               (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
47318       }
```

47319 **APPLICATION USAGE**

47320 Applications should use *sysconf*(_SC_CLK_TCK) to determine the number of clock ticks per
47321 second as it may vary from system to system.

47322 **RATIONALE**

47323 The accuracy of the times reported is intentionally left unspecified to allow implementations
47324 flexibility in design, from uniprocessor to multi-processor networks.

47325 The inclusion of times of child processes is recursive, so that a parent process may collect the
47326 total times of all of its descendants. But the times of a child are only added to those of its parent
47327 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process
47328 can always see the total times of all its descendants; see also the discussion of the term
47329 ''realtime'' in *alarm*( ).

47330 If the type **clock_t** is defined to be a signed 32-bit integer, it overflows in somewhat more than a
47331 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual
47332 systems that run continuously for longer than that. This volume of IEEE Std 1003.1-2001 permits
47333 an implementation to make the reference point for the returned value be the start-up time of the
47334 process, rather than system start-up time.

47335 The term ''charge'' in this context has nothing to do with billing for services. The operating
47336 system accounts for time used in this way. That information must be correct, regardless of how
47337 that information is used.

47338 **FUTURE DIRECTIONS**

47339 None.

47340 **SEE ALSO**

47341 *alarm*( ), *exec*, *fork*( ), *sysconf*( ), *time*( ), *wait*( ), the Base Definitions volume of
47342 IEEE Std 1003.1-2001, **<sys/times.h>**

47343 **CHANGE HISTORY**

47344 First released in Issue 1. Derived from Issue 1 of the SVID.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

47345 **NAME**

47346 timezone — difference from UTC and local standard time

47347 **SYNOPSIS**

47348 XSI
```
#include <time.h>
```

47349
```
extern long timezone;
```

47350

47351 **DESCRIPTION**

47352 Refer to *tzset*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

47353 **NAME**

47354 tmpfile — create a temporary file

47355 **SYNOPSIS**

47356
```
#include <stdio.h>
```

47357
```
FILE *tmpfile(void);
```

47358 **DESCRIPTION**

47359 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47360 conflict between the requirements described here and the ISO C standard is unintentional. This
47361 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47362 The *tmpfile*( ) function shall create a temporary file and open a corresponding stream. The file
47363 shall be automatically deleted when all references to the file are closed. The file is opened as in
47364 *fopen*( ) for update (*w+*).

47365 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile*( ) is
47366 killed while it is processing a call to *tmpfile*( ).

47367 An error message may be written to standard error if the stream cannot be opened.

47368 **RETURN VALUE**

47369 Upon successful completion, *tmpfile*( ) shall return a pointer to the stream of the file that is
47370 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

47371 **ERRORS**

47372 The *tmpfile*( ) function shall fail if:

47373 CX [EINTR] A signal was caught during *tmpfile*( ).

47374 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

47375 CX [ENFILE] The maximum allowable number of files is currently open in the system.

47376 CX [ENOSPC] The directory or file system which would contain the new file cannot be
47377 expanded.

47378 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly
47379 in an object of type **off_t**.

47380 The *tmpfile*( ) function may fail if:

47381 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

47382 CX [ENOMEM] Insufficient storage space is available.

47383 **EXAMPLES**

47384 **Creating a Temporary File**

47385 The following example creates a temporary file for update, and returns a pointer to a stream for
47386 the created file in the *fp* variable.

47387
```
#include <stdio.h>
```
47388
```
...
```
47389
```
FILE *fp;
```

47390
```
fp = tmpfile ();
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

47391 **APPLICATION USAGE**

47392 It should be possible to open at least {TMP_MAX} temporary files during the lifetime of the
47393 program (this limit may be shared with *tmpnam*( )) and there should be no limit on the number
47394 simultaneously open other than this limit and any limit on the number of open files
47395 ({FOPEN_MAX}).

47396 **RATIONALE**

47397 None.

47398 **FUTURE DIRECTIONS**

47399 None.

47400 **SEE ALSO**

47401 *fopen*( ), *tmpnam*( ), *unlink*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

47402 **CHANGE HISTORY**

47403 First released in Issue 1. Derived from Issue 1 of the SVID.

47404 **Issue 5**

47405 Large File Summit extensions are added.

47406 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
47407 in previous issues.

47408 **Issue 6**

47409 Extensions beyond the ISO C standard are marked.

47410 The following new requirements on POSIX implementations derive from alignment with the
47411 Single UNIX Specification:

47412 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
47413 large files.

47414 • The [EMFILE] optional error condition is added.

47415 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999
47416 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

tmpnam — create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tmpnam(char *s);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *tmpnam*( ) function shall generate a string that is a valid filename and that is not the same as the name of an existing file. The function is potentially capable of generating {TMP_MAX} different strings, but any or all of them may already be in use by existing files and thus not be suitable return values.

The *tmpnam*( ) function generates a different string each time it is called from the same process, up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behavior is implementation-defined.

The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *tmpnam*( ).

CX If the application uses any of the functions guaranteed to be available if either _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS is defined, the application shall ensure that the *tmpnam*( ) function is called with a non-NULL parameter.

RETURN VALUE

Upon successful completion, *tmpnam*( ) shall return a pointer to a string. If no suitable string can be generated, the *tmpnam*( ) function shall return a null pointer.

If the argument *s* is a null pointer, *tmpnam*( ) shall leave its result in an internal static object and return a pointer to that object. Subsequent calls to *tmpnam*( ) may modify the same object. If the argument *s* is not a null pointer, it is presumed to point to an array of at least L_tmpnam **char**s; *tmpnam*( ) shall write its result in that array and shall return the argument as its value.

ERRORS

No errors are defined.

EXAMPLES

Generating a Filename

The following example generates a unique filename and stores it in the array pointed to by *ptr*.

```
#include <stdio.h>
...
char filename[L_tmpnam+1];
char *ptr;

ptr = tmpnam(filename);
```

APPLICATION USAGE

This function only creates filenames. It is the application's responsibility to create and remove the files.

Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find *tmpfile*( ) more useful.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

47460 **RATIONALE**

47461 None.

47462 **FUTURE DIRECTIONS**

47463 None.

47464 **SEE ALSO**

47465 *fopen*( ), *open*( ), *tempnam*( ), *tmpfile*( ), *unlink*( ), the Base Definitions volume of
47466 IEEE Std 1003.1-2001, **<stdio.h>**

47467 **CHANGE HISTORY**

47468 First released in Issue 1. Derived from Issue 1 of the SVID.

47469 **Issue 5**

47470 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

47471 **Issue 6**

47472 Extensions beyond the ISO C standard are marked.

47473 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

47474 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

47475 **NAME**

47476 toascii — translate an integer to a 7-bit ASCII character

47477 **SYNOPSIS**

47478 XSI
```
#include <ctype.h>
```

47479
```
int toascii(int c);
```

47480

47481 **DESCRIPTION**

47482 The *toascii*( ) function shall convert its argument into a 7-bit ASCII character.

47483 **RETURN VALUE**

47484 The *toascii*( ) function shall return the value (*c* &0x7f).

47485 **ERRORS**

47486 No errors are returned.

47487 **EXAMPLES**

47488 None.

47489 **APPLICATION USAGE**

47490 None.

47491 **RATIONALE**

47492 None.

47493 **FUTURE DIRECTIONS**

47494 None.

47495 **SEE ALSO**

47496 *isascii*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<ctype.h>**

47497 **CHANGE HISTORY**

47498 First released in Issue 1. Derived from Issue 1 of the SVID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

tolower — transliterate uppercase characters to lowercase

SYNOPSIS

```
#include <ctype.h>

int tolower(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *tolower*( ) function has as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined. If the argument of *tolower*( ) represents an uppercase letter, and there exists a CX corresponding lowercase letter (as defined by character type information in the program locale category *LC_CTYPE*), the result shall be the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, *tolower*( ) shall return the lowercase letter corresponding to the argument passed; otherwise, it shall return the argument unchanged.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

toupper — transliterate lowercase characters to uppercase

SYNOPSIS

```
#include <ctype.h>

int toupper(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *toupper*( ) function has as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined. If the argument of *toupper*( ) represents a lowercase letter, and there exists a CX corresponding uppercase letter (as defined by character type information in the program locale category *LC_CTYPE*), the result shall be the corresponding uppercase letter. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, *toupper*( ) shall return the uppercase letter corresponding to the argument passed.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

47567 **NAME**

47568 towctrans — wide-character transliteration

47569 **SYNOPSIS**

47570
```
#include <wctype.h>
```

47571
```
wint_t towctrans(wint_t wc, wctrans_t desc);
```

47572 **DESCRIPTION**

47573 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47574 conflict between the requirements described here and the ISO C standard is unintentional. This
47575 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47576 The *towctrans*( ) function shall transliterate the wide-character code *wc* using the mapping
47577 described by *desc*. The current setting of the *LC_CTYPE* category should be the same as during
47578 CX the call to *wctrans*( ) that returned the value *desc*. If the value of *desc* is invalid (that is, not
47579 obtained by a call to *wctrans*( ) or *desc* is invalidated by a subsequent call to *setlocale*( ) that has
47580 affected category *LC_CTYPE*), the result is unspecified.

47581 An application wishing to check for error situations should set *errno* to 0 before calling
47582 *towctrans*( ). If *errno* is non-zero on return, an error has occurred.

47583 **RETURN VALUE**

47584 If successful, the *towctrans*( ) function shall return the mapped value of *wc* using the mapping
47585 described by *desc*. Otherwise, it shall return *wc* unchanged.

47586 **ERRORS**

47587 The *towctrans*( ) function may fail if:

47588 CX [EINVAL] *desc* contains an invalid transliteration descriptor.

47589 **EXAMPLES**

47590 None.

47591 **APPLICATION USAGE**

47592 The strings `"tolower"` and `"toupper"` are reserved for the standard mapping names. In the
47593 table below, the functions in the left column are equivalent to the functions in the right column.

47594
```
towlower(wc)    towctrans(wc, wctrans("tolower"))
```
47595
```
towupper(wc)    towctrans(wc, wctrans("toupper"))
```

47596 **RATIONALE**

47597 None.

47598 **FUTURE DIRECTIONS**

47599 None.

47600 **SEE ALSO**

47601 *towlower*( ), *towupper*( ), *wctrans*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
47602 **<wctype.h>**

47603 **CHANGE HISTORY**

47604 First released in Issue 5. Derived from ISO / IEC 9899: 1990 / Amendment 1: 1995 (E).

47605 **Issue 6**

47606 Extensions beyond the ISO C standard are marked.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

towlower — transliterate uppercase wide-character code to lowercase

SYNOPSIS

```
#include <wctype.h>

wint_t towlower(wint_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *towlower*( ) function has as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of *towlower*( ) represents an uppercase wide-character code, and there exists a corresponding lowercase wide-character code (as defined by character type information in the program locale category *LC_CTYPE*), the result shall be the corresponding lowercase wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, *towlower*( ) shall return the lowercase letter corresponding to the argument passed; otherwise, it shall return the argument unchanged.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<wctype.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

The following change has been made in this issue for alignment with ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

47649 **NAME**

47650 towupper — transliterate lowercase wide-character code to uppercase

47651 **SYNOPSIS**

47652 ```
#include <wctype.h>
```

47653 ```
wint_t towupper(wint_t wc);
```

47654 **DESCRIPTION**

47655 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47656 conflict between the requirements described here and the ISO C standard is unintentional. This
47657 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47658 The *towupper*( ) function has as a domain a type **wint_t**, the value of which the application shall
47659 ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a
47660 valid character in the current locale or the value of WEOF. If the argument has any other value,
47661 the behavior is undefined. If the argument of *towupper*( ) represents a lowercase wide-character
47662 code, and there exists a corresponding uppercase wide-character code (as defined by character
47663 type information in the program locale category *LC_CTYPE*), the result shall be the
47664 corresponding uppercase wide-character code. All other arguments in the domain are returned
47665 unchanged.

47666 **RETURN VALUE**

47667 Upon successful completion, *towupper*( ) shall return the uppercase letter corresponding to the
47668 argument passed. Otherwise, it shall return the argument unchanged.

47669 **ERRORS**

47670 No errors are defined.

47671 **EXAMPLES**

47672 None.

47673 **APPLICATION USAGE**

47674 None.

47675 **RATIONALE**

47676 None.

47677 **FUTURE DIRECTIONS**

47678 None.

47679 **SEE ALSO**

47680 *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<wctype.h>**,
47681 **<wchar.h>**

47682 **CHANGE HISTORY**

47683 First released in Issue 4.

47684 **Issue 5**

47685 The following change has been made in this issue for alignment with
47686 ISO / IEC 9899: 1990 / Amendment 1: 1995 (E):

47687 • The SYNOPSIS has been changed to indicate that this function and associated data types are
47688 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

47689 **Issue 6**

47690 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

trunc, truncf, truncl — round to truncated integer value

SYNOPSIS

```
#include <math.h>

double trunc(double x);
float truncf(float x);
long double truncl(long double x);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall round their argument to the integer value, in floating format, nearest to but no larger in magnitude than the argument.

RETURN VALUE

Upon successful completion, these functions shall return the truncated integer value.

MX If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

truncate — truncate a file to a specified length

SYNOPSIS

XSI
```
#include <unistd.h>

int truncate(const char *path, off_t length);
```

DESCRIPTION

The *truncate*( ) function shall cause the regular file named by *path* to have a size which shall be equal to *length* bytes.

If the file previously was larger than *length*, the extra data is discarded. If the file was previously shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

The application shall ensure that the process has write permission for the file.

If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the process.

This function shall not modify the file offset for any open file descriptions associated with the file. Upon successful completion, if the file size is changed, this function shall mark for update the *st_ctime* and *st_mtime* fields of the file, and the S_ISUID and S_ISGID bits of the file mode may be cleared.

RETURN VALUE

Upon successful completion, *truncate*( ) shall return 0. Otherwise, –1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *truncate*( ) function shall fail if:

[EINTR] A signal was caught during execution.

[EINVAL] The *length* argument was less than 0.

[EFBIG] or [EINVAL]
The *length* argument was greater than the maximum file size.

[EIO] An I/O error occurred while reading from or writing to a file system.

[EACCES] A component of the path prefix denies search permission, or write permission is denied on the file.

[EISDIR] The named file is a directory.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix of *path* is not a directory.

[EROFS] The named file resides on a read-only file system.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The *truncate*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

open( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Large File Summit extensions are added.

Issue 6

This reference page is split out from the *ftruncate*( ) reference page.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

47787 **NAME**

47788 truncf, truncl — round to truncated integer value

47789 **SYNOPSIS**

47790
```
#include <math.h>
```

47791
```
float truncf(float x);
```

47792
```
long double truncl(long double x);
```

47793 **DESCRIPTION**

47794 Refer to *trunc*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

47795 **NAME**

47796 tsearch — search a binary search tree

47797 **SYNOPSIS**

47798 XSI
```
#include <search.h>

void *tsearch(const void *key, void **rootp,
    int (*compar)(const void *, const void *));
```
47801

47802 **DESCRIPTION**

47803 Refer to *tdelete*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

47804 **NAME**

47805 ttyname, ttyname_r — find the pathname of a terminal

47806 **SYNOPSIS**

47807 ```
#include <unistd.h>
```

47808 ```
char *ttyname(int fildes);
```

47809 TSF ```
int ttyname_r(int fildes, char *name, size_t namesize);
```

47810

47811 **DESCRIPTION**

47812 The *ttyname*( ) function shall return a pointer to a string containing a null-terminated pathname
47813 of the terminal associated with file descriptor *fildes*. The return value may point to static data
47814 whose content is overwritten by each call.

47815 The *ttyname*( ) function need not be reentrant. A function that is not required to be reentrant is
47816 not required to be thread-safe.

47817 TSF The *ttyname_r*( ) function shall store the null-terminated pathname of the terminal associated
47818 with the file descriptor *fildes* in the character array referenced by *name*. The array is *namesize*
47819 characters long and should have space for the name and the terminating null character. The
47820 maximum length of the terminal name shall be {TTY_NAME_MAX}.

47821 **RETURN VALUE**

47822 Upon successful completion, *ttyname*( ) shall return a pointer to a string. Otherwise, a null
47823 pointer shall be returned and *errno* set to indicate the error.

47824 TSF If successful, the *ttyname_r*( ) function shall return zero. Otherwise, an error number shall be
47825 returned to indicate the error.

47826 **ERRORS**

47827 The *ttyname*( ) function may fail if:

47828 [EBADF] The *fildes* argument is not a valid file descriptor.

47829 [ENOTTY] The *fildes* argument does not refer to a terminal.

47830 The *ttyname_r*( ) function may fail if:

47831 TSF [EBADF] The *fildes* argument is not a valid file descriptor.

47832 TSF [ENOTTY] The *fildes* argument does not refer to a terminal.

47833 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned
47834 including the terminating null character.

47835 **EXAMPLES**

47836 None.

47837 **APPLICATION USAGE**

47838 None.

47839 **RATIONALE**

47840 The term ‘‘terminal’’ is used instead of the historical term ‘‘terminal device’’ in order to avoid a
47841 reference to an undefined term.

47842 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-
47843 zero value if it fails. The non-thread-safe version may return the name in a static data area that
47844 may be overwritten by each call.

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
```

47845 **FUTURE DIRECTIONS**

47846 None.

47847 **SEE ALSO**

47848 The Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

47849 **CHANGE HISTORY**

47850 First released in Issue 1. Derived from Issue 1 of the SVID.

47851 **Issue 5**

47852 The *ttyname_r*( ) function is included for alignment with the POSIX Threads Extension.

47853 A note indicating that the *ttyname*( ) function need not be reentrant is added to the
47854 DESCRIPTION.

47855 **Issue 6**

47856 The *ttyname_r*( ) function is marked as part of the Thread-Safe Functions option.

47857 The following new requirements on POSIX implementations derive from alignment with the
47858 Single UNIX Specification:

47859 - The statement that *errno* is set on error is added.

47860 - The [EBADF] and [ENOTTY] optional error conditions are added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

47861 **NAME**

47862 twalk — traverse a binary search tree

47863 **SYNOPSIS**

47864 XSI
```
#include <search.h>

void twalk(const void *root,
    void (*action)(const void *, VISIT, int ));
```
47867

47868 **DESCRIPTION**

47869 Refer to *tdelete*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

daylight, timezone, tzname, tzset — set timezone conversion information

SYNOPSIS

```
#include <time.h>

extern int daylight;
extern long timezone;
extern char *tzname[2];
void tzset(void);
```

DESCRIPTION

The *tzset*( ) function shall use the value of the environment variable *TZ* to set time conversion information used by *ctime*( ), *localtime*( ), *mktime*( ), and *strftime*( ). If *TZ* is absent from the environment, implementation-defined default timezone information shall be used.

The *tzset*( ) function shall set the external variable *tzname* as follows:

```
tzname[0] = "std";
tzname[1] = "dst";
```

where *std* and *dst* are as described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

The *tzset*( ) function also shall set the external variable *daylight* to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

RETURN VALUE

The *tzset*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

Example *TZ* variables and their timezone differences are given in the table below:

TZ	*timezone*
EST5EDT	5*60*60
GMT0	0*60*60
JST-9	–9*60*60
MET-1MEST	–1*60*60
MST7MDT	7*60*60
PST8PDT	8*60*60

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE ALSO

ctime( ), *localtime*( ), *mktime*( ), *strftime*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables, **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The example is corrected.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ualarm — set the interval timer

SYNOPSIS

OB XSI
```
#include <unistd.h>

useconds_t ualarm(useconds_t useconds, useconds_t interval);
```

DESCRIPTION

The *ualarm*( ) function shall cause the SIGALRM signal to be generated for the calling process after the number of realtime microseconds specified by the *useconds* argument has elapsed. When the *interval* argument is non-zero, repeated timeout notification occurs with a period in microseconds specified by the *interval* argument. If the notification signal, SIGALRM, is not caught or ignored, the calling process is terminated.

Implementations may place limitations on the granularity of timer values. For each interval timer, if the requested timer value requires a finer granularity than the implementation supports, the actual timer value shall be rounded up to the next supported value.

Interactions between *ualarm*( ) and any of the following are unspecified:

alarm( )
nanosleep( )
setitimer( )
timer_create( )
timer_delete( )
timer_getoverrun( )
timer_gettime( )
timer_settime( )
sleep( )

RETURN VALUE

The *ualarm*( ) function shall return the number of microseconds remaining from the previous *ualarm*( ) call. If no timeouts are pending or if *ualarm*( ) has not previously been called, *ualarm*( ) shall return 0.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

Applications are recommended to use *nanosleep*( ) if the Timers option is supported, or *setitimer*( ), *timer_create*( ), *timer_delete*( ), *timer_getoverrun*( ), *timer_gettime*( ), or *timer_settime*( ) instead of this function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *nanosleep*( ), *setitimer*( ), *sleep*( ), *timer_create*( ), *timer_delete*( ), *timer_getoverrun*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

47963 **CHANGE HISTORY**

47964 First released in Issue 4, Version 2.

47965 **Issue 5**

47966 Moved from X/OPEN UNIX extension to BASE.

47967 **Issue 6**

47968 This function is marked obsolescent.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ulimit — get and set process limits

SYNOPSIS

XSI
```
#include <ulimit.h>

long ulimit(int cmd, ...);
```

DESCRIPTION

The *ulimit*( ) function shall control process limits. The process limits that can be controlled by this function include the maximum size of a single file that can be written (this is equivalent to using *setrlimit*( ) with RLIMIT_FSIZE). The *cmd* values, defined in **<ulimit.h>**, include:

UL_GETFSIZE Return the file size limit (RLIMIT_FSIZE) of the process. The limit shall be in units of 512-byte blocks and shall be inherited by child processes. Files of any size can be read. The return value shall be the integer part of the soft file size limit divided by 512. If the result cannot be represented as a **long**, the result is unspecified.

UL_SETFSIZE Set the file size limit for output operations of the process to the value of the second argument, taken as a **long**, multiplied by 512. If the result would overflow an **rlim_t**, the actual value set is unspecified. Any process may decrease its own limit, but only a process with appropriate privileges may increase the limit. The return value shall be the integer part of the new file size limit divided by 512.

The *ulimit*( ) function shall not change the setting of *errno* if successful.

As all return values are permissible in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *ulimit*( ), and, if it returns –1, check to see if *errno* is non-zero.

RETURN VALUE

Upon successful completion, *ulimit*( ) shall return the value of the requested limit. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *ulimit*( ) function shall fail and the limit shall be unchanged if:

[EINVAL] The *cmd* argument is not valid.

[EPERM] A process not having appropriate privileges attempts to increase its file size limit.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48010 **SEE ALSO**

48011 *getrlimit*( ), *setrlimit*( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<ulimit.h>**

48012 **CHANGE HISTORY**

48013 First released in Issue 1. Derived from Issue 1 of the SVID.

48014 **Issue 5**

48015 In the description of UL_SETFSIZE, the text is corrected to refer to **rlim_t** rather than the
48016 spurious **rlimit_t**.

48017 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

umask — set and get the file mode creation mask

SYNOPSIS

```
#include <sys/stat.h>

mode_t umask(mode_t cmask);
```

DESCRIPTION

The *umask*( ) function shall set the process' file mode creation mask to *cmask* and return the previous value of the mask. Only the file permission bits of *cmask* (see **<sys/stat.h>**) are used; the meaning of the other bits is implementation-defined.

The process' file mode creation mask is used during *open*( ), *creat*( ), *mkdir*( ), and *mkfifo*( ) to turn off permission bits in the *mode* argument supplied. Bit positions that are set in *cmask* are cleared in the mode of the created file.

RETURN VALUE

The file permission bits in the value returned by *umask*( ) shall be the previous value of the file mode creation mask. The state of any other bits in that value is unspecified, except that a subsequent call to *umask*( ) with the returned value as *cmask* shall leave the state of the mask the same as its state before the first call, including any unspecified use of those bits.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Unsigned argument and return types for *umask*( ) were proposed. The return type and the argument were both changed to **mode_t**.

Historical implementations have made use of additional bits in *cmask* for their implementation-defined purposes. The addition of the text that the meaning of other bits of the field is implementation-defined permits these implementations to conform to this volume of IEEE Std 1003.1-2001.

FUTURE DIRECTIONS

None.

SEE ALSO

creat( ), *mkdir*( ), *mkfifo*( ), *open*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48057 The following new requirements on POSIX implementations derive from alignment with the
48058 Single UNIX Specification:

48059 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
48060 required for conforming implementations of previous POSIX specifications, it was not
48061 required for UNIX applications.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

uname — get the name of the current system

SYNOPSIS

```
#include <sys/utsname.h>

int uname(struct utsname *name);
```

DESCRIPTION

The *uname*( ) function shall store information identifying the current system in the structure pointed to by *name*.

The *uname*( ) function uses the **utsname** structure defined in **<sys/utsname.h>**.

The *uname*( ) function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

The format of each member is implementation-defined.

RETURN VALUE

Upon successful completion, a non-negative value shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

RATIONALE

The values of the structure members are not constrained to have any relation to the version of this volume of IEEE Std 1003.1-2001 implemented in the operating system. An application should instead depend on _POSIX_VERSION and related constants defined in **<unistd.h>**.

This volume of IEEE Std 1003.1-2001 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

The *uname*( ) function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

4.3 BSD has *gethostname*( ) and *gethostid*( ), which return a symbolic name and a numeric value, respectively. There are related *sethostname*( ) and *sethostid*( ) functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

FUTURE DIRECTIONS

None.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48104 **SEE ALSO**

48105 The Base Definitions volume of IEEE Std 1003.1-2001, **<sys/utsname.h>**

48106 **CHANGE HISTORY**

48107 First released in Issue 1. Derived from Issue 1 of the SVID.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

ungetc — push byte back into input stream

SYNOPSIS

```
#include <stdio.h>

int ungetc(int c, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *ungetc*( ) function shall push the byte specified by *c* (converted to an **unsigned char**) back onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by *stream*) to a file-positioning function (*fseek*( ), *fsetpos*( ), or *rewind*( )) shall discard any pushed-back bytes for the stream. The external storage corresponding to the stream shall be unchanged.

One byte of push-back shall be provided. If *ungetc*( ) is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall be left unchanged.

A successful call to *ungetc*( ) shall clear the end-of-file indicator for the stream. The value of the file-position indicator for the stream after reading or discarding all pushed-back bytes shall be the same as it was before the bytes were pushed back. The file-position indicator is decremented by each successful call to *ungetc*( ); if its value was 0 before a call, its value is unspecified after the call.

RETURN VALUE

Upon successful completion, *ungetc*( ) shall return the byte pushed back after conversion. Otherwise, it shall return EOF.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fseek( ), *getc*( ), *fsetpos*( ), *read*( ), *rewind*( ), *setbuf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

ungetwc — push wide-character code back into the input stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t ungetwc(wint_t wc, FILE *stream);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *ungetwc*( ) function shall push the character corresponding to the wide-character code specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters shall be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by *stream*) to a file-positioning function (*fseek*( ), *fsetpos*( ), or *rewind*( )) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

At least one character of push-back shall be provided. If *ungetwc*( ) is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream shall be left unchanged.

A successful call to *ungetwc*( ) shall clear the end-of-file indicator for the stream. The value of the file-position indicator for the stream after reading or discarding all pushed-back characters shall be the same as it was before the characters were pushed back. The file-position indicator is decremented (by one or more) by each successful call to *ungetwc*( ); if its value was 0 before a call, its value is unspecified after the call.

RETURN VALUE

Upon successful completion, *ungetwc*( ) shall return the wide-character code corresponding to the pushed-back character. Otherwise, it shall return WEOF.

ERRORS

The *ungetwc*( ) function may fail if:

CX [EILSEQ] An invalid character sequence is detected, or a wide-character code does not correspond to a valid character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE ALSO

fseek( ), *fsetpos*( ), *read*( ), *rewind*( ), *setbuf*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The Optional Header (OH) marking is removed from **<stdio.h>**.

Issue 6

The [EILSEQ] optional error condition is marked CX.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48200 **NAME**

48201 unlink — remove a directory entry

48202 **SYNOPSIS**

48203 `#include <unistd.h>`

48204 `int unlink(const char *path);`

48205 **DESCRIPTION**

48206 The *unlink*( ) function shall remove a link to a file. If *path* names a symbolic link, *unlink*( ) shall
48207 remove the symbolic link named by *path* and shall not affect any file or directory named by the
48208 contents of the symbolic link. Otherwise, *unlink*( ) shall remove the link named by the pathname
48209 pointed to by *path* and shall decrement the link count of the file referenced by the link.

48210 When the file's link count becomes 0 and no process has the file open, the space occupied by the
48211 file shall be freed and the file shall no longer be accessible. If one or more processes have the file
48212 open when the last link is removed, the link shall be removed before *unlink*( ) returns, but the
48213 removal of the file contents shall be postponed until all references to the file are closed.

48214 The *path* argument shall not name a directory unless the process has appropriate privileges and
48215 the implementation supports using *unlink*( ) on directories.

48216 Upon successful completion, *unlink*( ) shall mark for update the *st_ctime* and *st_mtime* fields of
48217 the parent directory. Also, if the file's link count is not 0, the *st_ctime* field of the file shall be
48218 marked for update.

48219 **RETURN VALUE**

48220 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
48221 indicate the error. If –1 is returned, the named file shall not be changed.

48222 **ERRORS**

48223 The *unlink*( ) function shall fail and shall not unlink the file if:

48224 [EACCES] Search permission is denied for a component of the path prefix, or write
48225 permission is denied on the directory containing the directory entry to be
48226 removed.

48227 [EBUSY] The file named by the *path* argument cannot be unlinked because it is being
48228 used by the system or another process and the implementation considers this
48229 an error.

48230 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
48231 argument.

48232 [ENAMETOOLONG]
48233 The length of the *path* argument exceeds {PATH_MAX} or a pathname
48234 component is longer than {NAME_MAX}.

48235 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48236 [ENOTDIR] A component of the path prefix is not a directory.

48237 [EPERM] The file named by *path* is a directory, and either the calling process does not
48238 have appropriate privileges, or the implementation prohibits using *unlink*( )
48239 on directories.

48240 XSI [EPERM] or [EACCES]
48241 The S_ISVTX flag is set on the directory containing the file referred to by the
48242 *path* argument and the caller is not the file owner, nor is the caller the
48243 directory owner, nor does the caller have appropriate privileges.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EROFS] The directory entry to be unlinked is part of a read-only file system.

The *unlink*( ) function may fail and not unlink the file if:

XSI [EBUSY] The file named by *path* is a named STREAM.

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *path* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

[ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.

EXAMPLES

Removing a Link to a File

The following example shows how to remove a link to a file named **/home/cnd/mod1** by removing the entry named **/modules/pass1**.

```
#include <unistd.h>

char *path = "/modules/pass1";
int   status;
...
status = unlink(path);
```

Checking for an Error

The following example fragment creates a temporary password lock file named **LOCKFILE**, which is defined as **/etc/ptmp**, and gets a file descriptor for it. If the file cannot be opened for writing, *unlink*( ) is used to remove the link between the file descriptor and **LOCKFILE**.

```
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>

#define LOCKFILE "/etc/ptmp"

int pfd;  /* Integer for file descriptor returned by open call. */
FILE *fpfd;  /* File pointer for use in putpwent(). */
...
/* Open password Lock file. If it exists, this is an error. */
if ((pfd = open(LOCKFILE, O_WRONLY| O_CREAT | O_EXCL, S_IRUSR
    | S_IWUSR | S_IRGRP | S_IROTH)) == -1)  {
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}

/* Lock file created; proceed with fdopen of lock file so that
   putpwent() can be used.
 */
if ((fpfd = fdopen(pfd, "w")) == NULL) {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
48287           close(pfd);
48288           unlink(LOCKFILE);
48289           exit(1);
48290       }
```

48291 **Replacing Files**

48292 The following example fragment uses *unlink*( ) to discard links to files, so that they can be
48293 replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error
48294 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can
48295 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```
48296       #include <sys/types.h>
48297       #include <stdio.h>
48298       #include <fcntl.h>
48299       #include <errno.h>
48300       #include <unistd.h>
48301       #include <sys/stat.h>

48302       #define LOCKFILE "/etc/ptmp"
48303       #define PASSWDFILE "/etc/passwd"
48304       #define SAVEFILE "/etc/opasswd"
48305       ...
48306       /* If no change was made, assume error and leave passwd unchanged. */
48307       if (!valid_change) {
48308           fprintf(stderr, "Could not change password for user %s\n", user);
48309           unlink(LOCKFILE);
48310           exit(1);
48311       }

48312       /* Change permissions on new password file. */
48313       chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

48314       /* Remove saved password file. */
48315       unlink(SAVEFILE);

48316       /* Save current password file. */
48317       link(PASSWDFILE, SAVEFILE);

48318       /* Remove current password file. */
48319       unlink(PASSWDFILE);

48320       /* Save new password file as current password file. */
48321       link(LOCKFILE,PASSWDFILE);

48322       /* Remove lock file. */
48323       unlink(LOCKFILE);

48324       exit(0);
```

48325 **APPLICATION USAGE**

48326 Applications should use *rmdir*( ) to remove a directory.

48327 **RATIONALE**

48328 Unlinking a directory is restricted to the superuser in many historical implementations for
48329 reasons given in *link*( ) (see also *rename*( )).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The meaning of [EBUSY] in historical implementations is ''mount point busy''. Since this volume of IEEE Std 1003.1-2001 does not cover the system administration concepts of mounting and unmounting, the description of the error was changed to ''resource busy''. (This meaning is used by some device drivers when a second process tries to open an exclusive use device.) The wording is also intended to allow implementations to refuse to remove a directory if it is the root or current working directory of any process.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *link*( ), *remove*( ), *rmdir*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The [EBUSY] error is added to the ''may fail'' part of the ERRORS section.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48354 **NAME**

48355 unlockpt — unlock a pseudo-terminal master/slave pair

48356 **SYNOPSIS**

48357 XSI
```
#include <stdlib.h>
```

48358
```
int unlockpt(int fildes);
```

48359

48360 **DESCRIPTION**

48361 The *unlockpt*( ) function shall unlock the slave pseudo-terminal device associated with the
48362 master to which *fildes* refers.

48363 Conforming applications shall ensure that they call *unlockpt*( ) before opening the slave side of a
48364 pseudo-terminal device.

48365 **RETURN VALUE**

48366 Upon successful completion, *unlockpt*( ) shall return 0. Otherwise, it shall return –1 and set *errno*
48367 to indicate the error.

48368 **ERRORS**

48369 The *unlockpt*( ) function may fail if:

48370 [EBADF] The *fildes* argument is not a file descriptor open for writing.

48371 [EINVAL] The *fildes* argument is not associated with a master pseudo-terminal device.

48372 **EXAMPLES**

48373 None.

48374 **APPLICATION USAGE**

48375 None.

48376 **RATIONALE**

48377 None.

48378 **FUTURE DIRECTIONS**

48379 None.

48380 **SEE ALSO**

48381 *grantpt*( ), *open*( ), *ptsname*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

48382 **CHANGE HISTORY**

48383 First released in Issue 4, Version 2.

48384 **Issue 5**

48385 Moved from X/OPEN UNIX extension to BASE.

48386 **Issue 6**

48387 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

unsetenv — remove an environment variable

SYNOPSIS

CX
```
#include <stdlib.h>

int unsetenv(const char *name);
```

DESCRIPTION

The *unsetenv*( ) function shall remove an environment variable from the environment of the calling process. The *name* argument points to a string, which is the name of the variable to be removed. The named argument shall not contain an '=' character. If the named variable does not exist in the current environment, the environment shall be unchanged and the function is considered to have completed successfully.

If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv*( ) is undefined. The *unsetenv*( ) function shall update the list of pointers to which *environ* points.

The *unsetenv*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, zero shall be returned. Otherwise, –1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

ERRORS

The *unsetenv*( ) function shall fail if:

[EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a string containing an '=' character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setenv*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

getenv( ), *setenv*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

usleep — suspend execution for an interval

SYNOPSIS

OB XSI
```
#include <unistd.h>

int usleep(useconds_t useconds);
```

DESCRIPTION

The *usleep*( ) function shall cause the calling thread to be suspended from execution until either the number of realtime microseconds specified by the argument *useconds* has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested due to the scheduling of other activity by the system.

The *useconds* argument shall be less than one million. If the value of *useconds* is 0, then the call has no effect.

If a SIGALRM signal is generated for the calling process during execution of *usleep*( ) and if the SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *usleep*( ) returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also unspecified whether it remains pending after *usleep*( ) returns or it is discarded.

If a SIGALRM signal is generated for the calling process during execution of *usleep*( ), except as a result of a prior call to *alarm*( ), and if the SIGALRM signal is not being ignored or blocked from delivery, it is unspecified whether that signal has any effect other than causing *usleep*( ) to return.

If a signal-catching function interrupts *usleep*( ) and examines or changes either the time a SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or whether the SIGALRM signal is blocked from delivery, the results are unspecified.

If a signal-catching function interrupts *usleep*( ) and calls *siglongjmp*( ) or *longjmp*( ) to restore an environment saved prior to the *usleep*( ) call, the action associated with the SIGALRM signal and the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored as part of the environment.

Implementations may place limitations on the granularity of timer values. For each interval timer, if the requested timer value requires a finer granularity than the implementation supports, the actual timer value shall be rounded up to the next supported value.

Interactions between *usleep*( ) and any of the following are unspecified:

nanosleep( )
setitimer( )
timer_create( )
timer_delete( )
timer_getoverrun( )
timer_gettime( )
timer_settime( )
ualarm( )
sleep( )

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48466 **RETURN VALUE**

48467 Upon successful completion, *usleep*( ) shall return 0; otherwise, it shall return –1 and set *errno* to
48468 indicate the error.

48469 **ERRORS**

48470 The *usleep*( ) function may fail if:

48471 [EINVAL] The time interval specified one million or more microseconds.

48472 **EXAMPLES**

48473 None.

48474 **APPLICATION USAGE**

48475 Applications are recommended to use *nanosleep*( ) if the Timers option is supported, or
48476 *setitimer*( ), *timer_create*( ), *timer_delete*( ), *timer_getoverrun*( ), *timer_gettime*( ), or *timer_settime*( )
48477 instead of this function.

48478 **RATIONALE**

48479 None.

48480 **FUTURE DIRECTIONS**

48481 None.

48482 **SEE ALSO**

48483 *alarm*( ), *getitimer*( ), *nanosleep*( ), *sigaction*( ), *sleep*( ), *timer_create*( ), *timer_delete*( ),
48484 *timer_getoverrun*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

48485 **CHANGE HISTORY**

48486 First released in Issue 4, Version 2.

48487 **Issue 5**

48488 Moved from X/OPEN UNIX extension to BASE.

48489 The DESCRIPTION is changed to indicate that timers are now thread-based rather than
48490 process-based.

48491 **Issue 6**

48492 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

48493 This function is marked obsolescent.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48494 **NAME**

48495 utime — set file access and modification times

48496 **SYNOPSIS**

48497 `#include <utime.h>`

48498 `int utime(const char *path, const struct utimbuf *times);`

48499 **DESCRIPTION**

48500 The *utime*( ) function shall set the access and modification times of the file named by the *path*
48501 argument.

48502 If *times* is a null pointer, the access and modification times of the file shall be set to the current
48503 time. The effective user ID of the process shall match the owner of the file, or the process has
48504 write permission to the file or has appropriate privileges, to use *utime*( ) in this manner.

48505 If *times* is not a null pointer, *times* shall be interpreted as a pointer to a **utimbuf** structure and the
48506 access and modification times shall be set to the values contained in the designated structure.
48507 Only a process with the effective user ID equal to the user ID of the file or a process with
48508 appropriate privileges may use *utime*( ) this way.

48509 The **utimbuf** structure is defined in the **<utime.h>** header. The times in the structure **utimbuf**
48510 are measured in seconds since the Epoch.

48511 Upon successful completion, *utime*( ) shall mark the time of the last file status change, *st_ctime*,
48512 to be updated; see **<sys/stat.h>**.

48513 **RETURN VALUE**

48514 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* shall
48515 be set to indicate the error, and the file times shall not be affected.

48516 **ERRORS**

48517 The *utime*( ) function shall fail if:

48518 [EACCES] Search permission is denied by a component of the path prefix; or the *times*
48519 argument is a null pointer and the effective user ID of the process does not
48520 match the owner of the file, the process does not have write permission for the
48521 file, and the process does not have appropriate privileges.

48522 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
48523 argument.

48524 [ENAMETOOLONG]
48525 The length of the *path* argument exceeds {PATH_MAX} or a pathname
48526 component is longer than {NAME_MAX}.

48527 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48528 [ENOTDIR] A component of the path prefix is not a directory.

48529 [EPERM] The *times* argument is not a null pointer and the calling process' effective user
48530 ID does not match the owner of the file and the calling process does not have
48531 the appropriate privileges.

48532 [EROFS] The file system containing the file is read-only.

48533 The *utime*( ) function may fail if:

48534 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
48535 resolution of the *path* argument.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48536 [ENAMETOOLONG]
48537 As a result of encountering a symbolic link in resolution of the *path* argument,
48538 the length of the substituted pathname string exceeded {PATH_MAX}.

48539 **EXAMPLES**
48540 None.

48541 **APPLICATION USAGE**
48542 None.

48543 **RATIONALE**
48544 The *actime* structure member must be present so that an application may set it, even though an
48545 implementation may ignore it and not change the access time on the file. If an application
48546 intends to leave one of the times of a file unchanged while changing the other, it should use
48547 *stat*( ) to retrieve the file's *st_atime* and *st_mtime* parameters, set *actime* and *modtime* in the buffer,
48548 and change one of them before making the *utime*( ) call.

48549 **FUTURE DIRECTIONS**
48550 None.

48551 **SEE ALSO**
48552 The Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<utime.h>**

48553 **CHANGE HISTORY**
48554 First released in Issue 1. Derived from Issue 1 of the SVID.

48555 **Issue 6**
48556 The following new requirements on POSIX implementations derive from alignment with the
48557 Single UNIX Specification:

48558 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
48559 required for conforming implementations of previous POSIX specifications, it was not
48560 required for UNIX applications.

48561 • The [ELOOP] mandatory error condition is added.

48562 • A second [ENAMETOOLONG] is added as an optional error condition.

48563 The following changes were made to align with the IEEE P1003.1a draft standard:

48564 • The [ELOOP] optional error condition is added.

48565 The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

utimes — set file access and modification times (**LEGACY**)

SYNOPSIS

XSI
```
#include <sys/time.h>

int utimes(const char *path, const struct timeval times[2]);
```

DESCRIPTION

The *utimes*( ) function shall set the access and modification times of the file pointed to by the *path* argument to the value of the *times* argument. The *utimes*( ) function allows time specifications accurate to the microsecond.

For *utimes*( ), the *times* argument is an array of **timeval** structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the **timeval** structure are measured in seconds and microseconds since the Epoch, although rounding toward the nearest second may occur.

If the *times* argument is a null pointer, the access and modification times of the file shall be set to the current time. The effective user ID of the process shall match the owner of the file, or has write access to the file or appropriate privileges to use this call in this manner. Upon completion, *utimes*( ) shall mark the time of the last file status change, *st_ctime*, for update.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* shall be set to indicate the error, and the file times shall not be affected.

ERRORS

The *utimes*( ) function shall fail if:

[EACCES] Search permission is denied by a component of the path prefix; or the *times* argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG]
The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

[EPERM] The *times* argument is not a null pointer and the calling process' effective user ID has write access to the file but does not match the owner of the file and the calling process does not have the appropriate privileges.

[EROFS] The file system containing the file is read-only.

The *utimes*( ) function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48609 **EXAMPLES**

48610 None.

48611 **APPLICATION USAGE**

48612 For applications portability, the *utime*() function should be used to set file access and
48613 modification times instead of *utimes*().

48614 **RATIONALE**

48615 None.

48616 **FUTURE DIRECTIONS**

48617 This function may be withdrawn in a future version.

48618 **SEE ALSO**

48619 *utime*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/time.h>**

48620 **CHANGE HISTORY**

48621 First released in Issue 4, Version 2.

48622 **Issue 5**

48623 Moved from X/OPEN UNIX extension to BASE.

48624 **Issue 6**

48625 This function is marked LEGACY.

48626 The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

48627 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
48628 [ELOOP] error condition is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48629 **NAME**

48630 va_arg, va_copy, va_end, va_start — handle variable argument list

48631 **SYNOPSIS**

48632 ```
#include <stdarg.h>
```

48633 ```
type va_arg(va_list ap, type);
```
48634 ```
void va_copy(va_list dest, va_list src);
```
48635 ```
void va_end(va_list ap);
```
48636 ```
void va_start(va_list ap, argN);
```

48637 **DESCRIPTION**

48638 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, **<stdarg.h>**.


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
```

NAME

vfork — create a new process; share virtual memory

SYNOPSIS

OB XSI
```
#include <unistd.h>

pid_t vfork(void);
```

DESCRIPTION

The *vfork*( ) function shall be equivalent to *fork*( ), except that the behavior is undefined if the process created by *vfork*( ) either modifies any data other than a variable of type **pid_t** used to store the return value from *vfork*( ), or returns from the function in which *vfork*( ) was called, or calls any other function before successfully calling *_exit*( ) or one of the *exec* family of functions.

RETURN VALUE

Upon successful completion, *vfork*( ) shall return 0 to the child process and return the process ID of the child process to the parent process. Otherwise, –1 shall be returned to the parent, no child process shall be created, and *errno* shall be set to indicate the error.

ERRORS

The *vfork*( ) function shall fail if:

[EAGAIN] The system-wide limit on the total number of processes under execution would be exceeded, or the system-imposed limit on the total number of processes under execution by a single user would be exceeded.

[ENOMEM] There is insufficient swap space for the new process.

EXAMPLES

None.

APPLICATION USAGE

Conforming applications are recommended not to depend on *vfork*( ), but to use *fork*( ) instead. The *vfork*( ) function may be withdrawn in a future version.

On some implementations, *vfork*( ) is equivalent to *fork*( ).

The *vfork*( ) function differs from *fork*( ) only in that the child process can share code and data with the calling process (parent process). This speeds cloning activity significantly at a risk to the integrity of the parent process if *vfork*( ) is misused.

The use of *vfork*( ) for any purpose except as a prelude to an immediate call to a function from the *exec* family, or to *_exit*( ), is not advised.

The *vfork*( ) function can be used to create new processes without fully copying the address space of the old process. If a forked process is simply going to call *exec*, the data space copied from the parent to the child by *fork*( ) is not used. This is particularly inefficient in a paged environment, making *vfork*( ) particularly useful. Depending upon the size of the parent's data space, *vfork*( ) can give a significant performance improvement over *fork*( ).

The *vfork*( ) function can normally be used just like *fork*( ). It does not work, however, to return while running in the child's context from the caller of *vfork*( ) since the eventual return from *vfork*( ) would then return to a no longer existent stack frame. Care should be taken, also, to call *_exit*( ) rather than *exit*( ) if *exec* cannot be used, since *exit*( ) flushes and closes standard I/O channels, thereby damaging the parent process' standard I/O data structures. (Even with *fork*( ), it is wrong to call *exit*( ), since buffered data would then be flushed twice.)

If signal handlers are invoked in the child process after *vfork*( ), they must follow the same rules as other code in the child process.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48684 **RATIONALE**

48685 None.

48686 **FUTURE DIRECTIONS**

48687 This function may be withdrawn in a future version.

48688 **SEE ALSO**

48689 *exec*, *exit*( ), *fork*( ), *wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

48690 **CHANGE HISTORY**

48691 First released in Issue 4, Version 2.

48692 **Issue 5**

48693 Moved from X/OPEN UNIX extension to BASE.

48694 **Issue 6**

48695 This function is marked obsolescent.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vfprintf(FILE *restrict stream, const char *restrict format,
    va_list ap);
int vprintf(const char *restrict format, va_list ap);
int vsnprintf(char *restrict s, size_t n, const char *restrict format,
    va_list ap);
int vsprintf(char *restrict s, const char *restrict format, va_list ap);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *vprintf*( ), *vfprintf*( ), *vsnprintf*( ), and *vsprintf*( ) functions shall be equivalent to *printf*( ), *fprintf*( ), *snprintf*( ), and *sprintf*( ) respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **<stdarg.h>**.

These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fprintf*( ).

ERRORS

Refer to *fprintf*( ).

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdarg.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The *vsnprintf*( ) function is added.

Issue 6

The *vfprintf*( ), *vprintf*( ), *vsnprintf*( ), and *vsprintf*( ) functions are updated for alignment with the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

vfscanf, vscanf, vsscanf — format input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vfscanf(FILE *restrict stream, const char *restrict format,
    va_list arg);
int vscanf(const char *restrict format, va_list arg);
int vsscanf(const char *restrict s, const char *restrict format,
    va_list arg);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *vscanf*( ), *vfscanf*( ), and *vsscanf*( ) functions shall be equivalent to the *scanf*( ), *fscanf*( ), and *sscanf*( ) functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header. These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fscanf*( ).

ERRORS

Refer to *fscanf*( ).

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdarg.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
    va_list arg);
int vswprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, va_list arg);
int vwprintf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *vfwprintf*( ), *vswprintf*( ), and *vwprintf*( ) functions shall be equivalent to *fwprintf*( ), *swprintf*( ), and *wprintf*( ) respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **<stdarg.h>**.

These functions shall not invoke the *va_end* macro. However, as these functions do invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fwprintf*( ).

ERRORS

Refer to *fwprintf*( ).

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fwprintf( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The *vfwprintf*( ), *vswprintf*( ), and *vwprintf*( ) prototypes are updated for alignment with the ISO/IEC 9899:1999 standard. ( )

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,
    va_list arg);
int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
    va_list arg);
int vwscanf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *vfwscanf*(), *vswscanf*(), and *vwscanf*() functions shall be equivalent to the *fwscanf*(), *swscanf*(), and *wscanf*() functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header. These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fwscanf*().

ERRORS

Refer to *fwscanf*().

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fwscanf(), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

48850 **NAME**

48851 vprintf — format the output of a stdarg argument list

48852 **SYNOPSIS**

48853 `#include <stdarg.h>`

48854 `#include <stdio.h>`

48855 `int vprintf(const char *restrict format, va_list ap);`

48856 **DESCRIPTION**

48857 Refer to *vfprintf*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48858 **NAME**

48859 vscanf — format input of a stdarg argument list

48860 **SYNOPSIS**

48861 `#include <stdarg.h>`

48862 `#include <stdio.h>`

48863 `int vscanf(const char *restrict format, va_list arg);`

48864 **DESCRIPTION**

48865 Refer to *vfscanf*( ).

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48866 **NAME**

48867 vsnprintf, vsprintf — format output of a stdarg argument list

48868 **SYNOPSIS**

48869 `#include <stdarg.h>`
48870 `#include <stdio.h>`

48871 `int vsnprintf(char *restrict s, size_t n,`
48872 `const char *restrict format, va_list ap);`
48873 `int vsprintf(char *restrict s, const char *restrict format,`
48874 `va_list ap);`

48875 **DESCRIPTION**

48876 Refer to *vfprintf*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48877 **NAME**

48878 vsscanf — format input of a stdarg argument list

48879 **SYNOPSIS**

48880 `#include <stdarg.h>`
48881 `#include <stdio.h>`

48882 `int vsscanf(const char *restrict s, const char *restrict format,`
48883 `va_list arg);`

48884 **DESCRIPTION**

48885 Refer to *vfscanf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48886 **NAME**

48887 vswprintf — wide-character formatted output of a stdarg argument list

48888 **SYNOPSIS**

48889
```
#include <stdarg.h>
```
48890
```
#include <stdio.h>
```
48891
```
#include <wchar.h>
```

48892
```
int vswprintf(wchar_t *restrict ws, size_t n,
```
48893
```
    const wchar_t *restrict format, va_list arg);
```

48894 **DESCRIPTION**

48895 Refer to *vfwprintf*( ).

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48896 **NAME**

48897 vswscanf — wide-character formatted input of a stdarg argument list

48898 **SYNOPSIS**

48899
```
#include <stdarg.h>
```
48900
```
#include <stdio.h>
```
48901
```
#include <wchar.h>
```

48902
```
int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
```
48903
```
    va_list arg);
```

48904 **DESCRIPTION**

48905 Refer to *vfwscanf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

48906 **NAME**

48907 vwprintf — wide-character formatted output of a stdarg argument list

48908 **SYNOPSIS**

48909 `#include <stdarg.h>`

48910 `#include <stdio.h>`

48911 `#include <wchar.h>`

48912 `int vwprintf(const wchar_t *restrict format, va_list arg);`

48913 **DESCRIPTION**

48914 Refer to *vfwprintf*( ).

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48915 **NAME**

48916 vwscanf — wide-character formatted input of a stdarg argument list

48917 **SYNOPSIS**

48918 ```
#include <stdarg.h>
```

48919 ```
#include <stdio.h>
```

48920 ```
#include <wchar.h>
```

48921 ```
int vwscanf(const wchar_t *restrict format, va_list arg);
```

48922 **DESCRIPTION**

48923 Refer to *vfwscanf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wait, waitpid — wait for a child process to stop or terminate

SYNOPSIS

```
#include <sys/wait.h>

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

DESCRIPTION

The *wait*( ) and *waitpid*( ) functions shall obtain status information pertaining to one of the caller's child processes. Various options permit status information to be obtained for child processes that have terminated or stopped. If status information is available for two or more child processes, the order in which their status is reported is unspecified.

The *wait*( ) function shall suspend execution of the calling thread until status information for one of the terminated child processes of the calling process is available, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. If more than one thread is suspended in *wait*( ) or *waitpid*( ) awaiting termination of the same process, exactly one thread shall return the process status at the time of the target process termination. If status information is available prior to the call to *wait*( ), return shall be immediate.

The *waitpid*( ) function shall be equivalent to *wait*( ) if the *pid* argument is (**pid_t**)–1 and the *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and *options* arguments.

The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid*( ) function shall only return the status of a child process from this set:

- If *pid* is equal to (**pid_t**)–1, *status* is requested for any child process. In this respect, *waitpid*( ) is then equivalent to *wait*( ).
- If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is requested.
- If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of the calling process.
- If *pid* is less than (**pid_t**)–1, *status* is requested for any child process whose process group ID is equal to the absolute value of *pid*.

The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the following flags, defined in the **<sys/wait.h>** header:

XSI WCONTINUED — The *waitpid*( ) function shall report the status of any continued child process specified by *pid* whose status has not been reported since it continued from a job control stop.

WNOHANG — The *waitpid*( ) function shall not suspend execution of the calling thread if *status* is not immediately available for one of the child processes specified by *pid*.

WUNTRACED — The status of any child processes specified by *pid* that are stopped, and whose status has not yet been reported since they stopped, shall also be reported to the requesting process.

XSI If the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, and the process has no unwaited-for children that were transformed into zombie processes, the calling thread shall block until all of the children of the process containing the calling thread terminate, and *wait*( ) and *waitpid*( ) shall fail and set *errno* to [ECHILD].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If *wait*( ) or *waitpid*( ) return because the status of a child process is available, these functions shall return a value equal to the process ID of the child process. In this case, if the value of the argument *stat_loc* is not a null pointer, information shall be stored in the location pointed to by *stat_loc*. The value stored at the location pointed to by *stat_loc* shall be 0 if and only if the status returned is from a terminated child process that terminated by one of the following means:

1. The process returned 0 from *main*( ).
2. The process called *_exit*( ) or *exit*( ) with a *status* argument of 0.
3. The process was terminated because the last thread in the process terminated.

Regardless of its value, this information may be interpreted using the following macros, which are defined in **<sys/wait.h>** and evaluate to integral expressions; the *stat_val* argument is the integer value pointed to by *stat_loc*.

WIFEXITED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that terminated normally.

WEXITSTATUS(*stat_val*)
: If the value of WIFEXITED(*stat_val*) is non-zero, this macro evaluates to the low-order 8 bits of the *status* argument that the child process passed to *_exit*( ) or *exit*( ), or the value the child process returned from *main*( ).

WIFSIGNALED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that terminated due to the receipt of a signal that was not caught (see **<signal.h>**).

WTERMSIG(*stat_val*)
: If the value of WIFSIGNALED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.

WIFSTOPPED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that is currently stopped.

WSTOPSIG(*stat_val*)
: If the value of WIFSTOPPED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.

XSI WIFCONTINUED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that has continued from a job control stop.

SPN It is unspecified whether the *status* value returned by calls to *wait*( ) or *waitpid*( ) for processes created by *posix_spawn*( ) or *posix_spawnp*( ) can indicate a WIFSTOPPED(*stat_val*) before subsequent calls to *wait*( ) or *waitpid*( ) indicate WIFEXITED(*stat_val*) as the result of an error detected before the new process image starts executing.

It is unspecified whether the *status* value returned by calls to *wait*( ) or *waitpid*( ) for processes created by *posix_spawn*( ) or *posix_spawnp*( ) can indicate a WIFSIGNALED(*stat_val*) if a signal is sent to the parent's process group after *posix_spawn*( ) or *posix_spawnp*( ) is called.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that specified the XSI WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFSTOPPED(**stat_loc*) shall evaluate to a non-zero value.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the information pointed to by *stat_loc* was stored by a call to *waitpid*() that specified the WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), WIFSTOPPED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*() that did not specify the WUNTRACED or WCONTINUED flags, or by a call to the *wait*() function, exactly one of the macros WIFEXITED(**stat_loc*) and WIFSIGNALED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*() that did not specify the WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait*() function, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If _POSIX_REALTIME_SIGNALS is defined, and the implementation queues the SIGCHLD signal, then if *wait*() or *waitpid*() returns because the status of a child process is available, any pending SIGCHLD signal associated with the process ID of the child process shall be discarded. Any other pending SIGCHLD signals shall remain pending.

Otherwise, if SIGCHLD is blocked, if *wait*() or *waitpid*() return because the status of a child process is available, any pending SIGCHLD signal shall be cleared unless the status of another child process is available.

For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD signal is delivered.

There may be additional implementation-defined circumstances under which *wait*() or *waitpid*() report *status*. This shall not occur unless the calling process or one of its child processes explicitly makes use of a non-standard extension. In these cases the interpretation of the reported *status* is implementation-defined.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes shall be assigned a new parent process ID corresponding to an implementation-defined system process.

RETURN VALUE

If *wait*() or *waitpid*() returns because the status of a child process is available, these functions shall return a value equal to the process ID of the child process for which *status* is reported. If *wait*() or *waitpid*() returns due to the delivery of a signal to the calling process, –1 shall be returned and *errno* set to [EINTR]. If *waitpid*() was invoked with WNOHANG set in *options*, it has at least one child process specified by *pid* for which *status* is not available, and *status* is not available for any process specified by *pid*, 0 is returned. Otherwise, (**pid_t**)–1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *wait*() function shall fail if:

[ECHILD] The calling process has no existing unwaited-for child processes.

[EINTR] The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.

The *waitpid*() function shall fail if:

[ECHILD] The process specified by *pid* does not exist or is not a child of the calling process, or the process group specified by *pid* does not exist or does not have any member process that is a child of the calling process.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[EINTR] The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.

[EINVAL] The *options* argument is not valid.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A call to the *wait*( ) or *waitpid*( ) function only returns *status* on an immediate child process of the calling process; that is, a child that was produced by a single *fork*( ) call (perhaps followed by an *exec* or other function calls) from the parent. If a child produces grandchildren by further use of *fork*( ), none of those grandchildren nor any of their descendants affect the behavior of a *wait*( ) from the original parent process. Nothing in this volume of IEEE Std 1003.1-2001 prevents an implementation from providing extensions that permit a process to get *status* from a grandchild or any other process, but a process that does not use such extensions must be guaranteed to see *status* from only its direct children.

The *waitpid*( ) function is provided for three reasons:

1. To support job control
2. To permit a non-blocking version of the *wait*( ) function
3. To permit a library routine, such as *system*( ) or *pclose*( ), to wait for its children without interfering with other terminated children for which the process has not waited

The first two of these facilities are based on the *wait3*( ) function provided by 4.3 BSD. The function uses the *options* argument, which is equivalent to an argument to *wait3*( ). The WUNTRACED flag is used only in conjunction with job control on systems supporting job control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped processes in that implementation: processes being traced via the *ptrace*( ) debugging facility and (untraced) processes stopped by job control signals. Since *ptrace*( ) is not part of this volume of IEEE Std 1003.1-2001, only the second type is relevant. The name WUNTRACED was retained because its usage is the same, even though the name is not intuitively meaningful in this context.

The third reason for the *waitpid*( ) function is to permit independent sections of a process to spawn and wait for children without interfering with each other. For example, the following problem occurs in developing a portable shell, or command interpreter:

```
stream = popen("/bin/true");
(void) system("sleep 100");
(void) pclose(stream);
```

On all historical implementations, the final *pclose*( ) fails to reap the *wait*( ) *status* of the *popen*( ).

The status values are retrieved by macros, rather than given as specific bit encodings as they are in most historical implementations (and thus expected by existing programs). This was necessary to eliminate a limitation on the number of signals an implementation can support that was inherent in the traditional encodings. This volume of IEEE Std 1003.1-2001 does require that a *status* value of zero corresponds to a process calling *_exit*(0), as this is the most common encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

These macros syntactically operate on an arbitrary integer value. The behavior is undefined unless that value is one stored by a successful call to *wait*( ) or *waitpid*( ) in the location pointed to by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other specifications in this volume of IEEE Std 1003.1-2001 or traditional usage. It also could have implied that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or passed as an argument to other functions before being interpreted by these macros.

The extension that affects *wait*( ) and *waitpid*( ) and is common in historical implementations is the *ptrace*( ) function. It is called by a child process and causes that child to stop and return a *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace*( ) children is traditionally returned regardless of the WUNTRACED flag (or by the *wait*( ) function). Most applications do not need to concern themselves with such extensions because they have control over what extensions they or their children use. However, applications, such as command interpreters, that invoke arbitrary processes may see this behavior when those arbitrary processes misuse such extensions.

Implementations that support **core** file creation or other implementation-defined actions on termination of some processes traditionally provide a bit in the *status* returned by *wait*( ) to indicate that such actions have occurred.

Allowing the *wait*( ) family of functions to discard a pending SIGCHLD signal that is associated with a successfully waited-for child process puts them into the *sigwait*( ) and *sigwaitinfo*( ) category with respect to SIGCHLD.

This definition allows implementations to treat a pending SIGCHLD signal as accepted by the process in *wait*( ), with the same meaning of ‘‘accepted’’ as when that word is applied to the *sigwait*( ) family of functions.

Allowing the *wait*( ) family of functions to behave this way permits an implementation to be able to deal precisely with SIGCHLD signals.

In particular, an implementation that does accept (discard) the SIGCHLD signal can make the following guarantees regardless of the queuing depth of signals in general (the list of waitable children can hold the SIGCHLD queue):

1. If a SIGCHLD signal handler is established via *sigaction*( ) without the SA_RESETHAND flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will be delivered to or accepted by the process for every child process that terminates.
2. A single *wait*( ) issued from a SIGCHLD signal handler can be guaranteed to return immediately with status information for a child process.
3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to receive a non-NULL pointer to a **siginfo_t** structure that describes a child process for which a wait via *waitpid*( ) or *waitid*( ) will not block or fail.
4. The *system*( ) function will not cause a process' SIGCHLD handler to be called as a result of the *fork*( )/*exec* executed within *system*( ) because *system*( ) will accept the SIGCHLD signal when it performs a *waitpid*( ) for its child process. This is a desirable behavior of *system*( ) so that it can be used in a library without causing side effects to the application linked with the library.

An implementation that does not permit the *wait*( ) family of functions to accept (discard) a pending SIGCHLD signal associated with a successfully waited-for child, cannot make the guarantees described above for the following reasons:

Guarantee #1

Although it might be assumed that reliable queuing of all SIGCHLD signals generated by the system can make this guarantee, the counter-example is the case of a process that blocks SIGCHLD and performs an indefinite loop of *fork*( )/*wait*( ) operations. If the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

implementation supports queued signals, then eventually the system will run out of memory for the queue. The guarantee cannot be made because there must be some limit to the depth of queuing.

Guarantees #2 and #3

These cannot be guaranteed unless the *wait*() family of functions accepts the SIGCHLD signal. Otherwise, a *fork*()/*wait*() executed while SIGCHLD is blocked (as in the *system*() function) will result in an invocation of the handler when SIGCHLD is unblocked, after the process has disappeared.

Guarantee #4

Although possible to make this guarantee, *system*() would have to set the SIGCHLD handler to SIG_DFL so that the SIGCHLD signal generated by its *fork*() would be discarded (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This would have the undesirable side effect of discarding all SIGCHLD signals pending to the process.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit*(), *fork*(), *waitid*(), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<sys/wait.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

The semantics of WIFSTOPPED(*stat_val*), WIFEXITED(*stat_val*), and WIFSIGNALED(*stat_val*) are defined with respect to *posix_spawn*() or *posix_spawnp*() for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

waitid — wait for a child process to change state

SYNOPSIS

XSI
```
#include <sys/wait.h>

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

DESCRIPTION

The *waitid*( ) function shall suspend the calling thread until one child of the process containing the calling thread changes state. It records the current state of a child in the structure pointed to by *infop*. If a child process changed state prior to the call to *waitid*( ), *waitid*( ) shall return immediately. If more than one thread is suspended in *wait*( ) or *waitpid*( ) waiting for termination of the same process, exactly one thread shall return the process status at the time of the target process termination.

The *idtype* and *id* arguments are used to specify which children *waitid*( ) waits for.

If *idtype* is P_PID, *waitid*( ) shall wait for the child with a process ID equal to (**pid_t**)*id*.

If *idtype* is P_PGID, *waitid*( ) shall wait for any child with a process group ID equal to (**pid_t**)*id*.

If *idtype* is P_ALL, *waitid*( ) shall wait for any children and *id* is ignored.

The *options* argument is used to specify which state changes *waitid*( ) shall wait for. It is formed by OR'ing together one or more of the following flags:

WEXITED	Wait for processes that have exited.
WSTOPPED	Status shall be returned for any child that has stopped upon receipt of a signal.
WCONTINUED	Status shall be returned for any child that was stopped and has been continued.
WNOHANG	Return immediately if there are no children to wait for.
WNOWAIT	Keep the process whose status is returned in *infop* in a waitable state. This shall not affect the state of the process; the process may be waited for again after this call completes.

The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid*( ) returns because a child process was found that satisfied the conditions indicated by the arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the system with the status of the process. The *si_signo* member shall always be equal to SIGCHLD.

RETURN VALUE

If WNOHANG was specified and there are no children to wait for, 0 shall be returned. If *waitid*( ) returns due to the change of state of one of its children, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

The *waitid*( ) function shall fail if:

[ECHILD]	The calling process has no existing unwaited-for child processes.
[EINTR]	The *waitid*( ) function was interrupted by a signal.
[EINVAL]	An invalid value was specified for *options*, or *idtype* and *id* specify an invalid set of processes.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit*( ), *wait*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/wait.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

49241 **NAME**

49242 waitpid — wait for a child process to stop or terminate

49243 **SYNOPSIS**

49244 `#include <sys/wait.h>`

49245 `pid_t waitpid(pid_t` *`pid`*`, int *`*`stat_loc`*`, int` *`options`*`);`

49246 **DESCRIPTION**

49247 Refer to *wait*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcrtomb — convert a wide-character code to a character (restartable)

SYNOPSIS

```
#include <stdio.h>

size_t wcrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

If *s* is a null pointer, the *wcrtomb*( ) function shall be equivalent to the call:

```
wcrtomb(buf, L'\0', ps)
```

where *buf* is an internal buffer.

If *s* is not a null pointer, the *wcrtomb*( ) function shall determine the number of bytes needed to represent the character that corresponds to the wide character given by *wc* (including any shift sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At most {MB_CUR_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state. The resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcrtomb*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *wcrtomb*( ).

CX If the application uses any of the _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS functions, the application shall ensure that the *wcrtomb*( ) function is called with a non-NULL *ps* argument.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *wcrtomb*( ) function shall return the number of bytes stored in the array object (including any shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)–1; the conversion state shall be undefined.

ERRORS

The *wcrtomb*( ) function may fail if:

CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

[EILSEQ] Invalid wide-character code is detected.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

In the DESCRIPTION, a note on using this function in a threaded application is added.

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The *wcrtomb*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcscat — concatenate two wide-character strings

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcscat*( ) function shall append a copy of the wide-character string pointed to by *ws2* (including the terminating null wide-character code) to the end of the wide-character string pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wcscat*( ) function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsncat( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, *s1* is changed to *ws1*.

The *wcscat*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcschr — wide-character string scanning operation

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcschr*( ) function shall locate the first occurrence of *wc* in the wide-character string pointed to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide-character string.

RETURN VALUE

Upon completion, *wcschr*( ) shall return a pointer to the wide-character code, or a null pointer if the wide-character code is not found.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsrchr( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcscmp — compare two wide-character strings

SYNOPSIS

```
#include <wchar.h>

int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcscmp*( ) function shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

RETURN VALUE

Upon completion, *wcscmp*( ) shall return an integer greater than, equal to, or less than 0, if the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsncmp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcscoll — wide-character string comparison using collating information

SYNOPSIS

```
#include <wchar.h>

int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcscoll*( ) function shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*, both interpreted as appropriate to the *LC_COLLATE* category of the current locale.

CX The *wcscoll*( ) function shall not change the setting of *errno* if successful.

An application wishing to check for error situations should set *errno* to 0 before calling *wcscoll*( ). If *errno* is non-zero on return, an error has occurred.

RETURN VALUE

Upon successful completion, *wcscoll*( ) shall return an integer greater than, equal to, or less than 0, according to whether the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, when both are interpreted as appropriate CX to the current locale. On error, *wcscoll*( ) shall set *errno*, but no return value is reserved to indicate an error.

ERRORS

The *wcscoll*( ) function may fail if:

CX [EINVAL] The *ws1* or *ws2* arguments contain wide-character codes outside the domain of the collating sequence.

EXAMPLES

None.

APPLICATION USAGE

The *wcsxfrm*( ) and *wcscmp*( ) functions should be used for sorting large lists.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscmp( ), *wcsxfrm*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcscpy — copy a wide-character string

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcscpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcscpy*( ) function shall copy the wide-character string pointed to by *ws2* (including the terminating null wide-character code) into the array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wcscpy*( ) function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsncpy( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The *wcscpy*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

wcscspn — get the length of a complementary wide substring

SYNOPSIS

```
#include <wchar.h>

size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcscspn*( ) function shall compute the length (in wide characters) of the maximum initial segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character codes *not* from the wide-character string pointed to by *ws2*.

RETURN VALUE

The *wcscspn*( ) function shall return the length of the initial substring of *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsspn( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The RETURN VALUE section is updated to indicate that *wcscspn*( ) returns the length of *ws1*, rather than *ws1* itself.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcsftime — convert date and time to a wide-character string

SYNOPSIS

```
#include <wchar.h>

size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,
    const wchar_t *restrict format, const struct tm *restrict timeptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsftime*( ) function shall be equivalent to the *strftime*( ) function, except that:

- The argument *wcs* points to the initial element of an array of wide characters into which the generated output is to be placed.
- The argument *maxsize* indicates the maximum number of wide characters to be placed in the output array.
- The argument *format* is a wide-character string and the conversion specifications are replaced by corresponding sequences of wide characters.
- The return value indicates the number of wide characters placed in the output array.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

If the total number of resulting wide-character codes including the terminating null wide-character code is no more than *maxsize*, *wcsftime*( ) shall return the number of wide-character codes placed into the array pointed to by *wcs*, not including the terminating null wide-character code. Otherwise, zero is returned and the contents of the array are unspecified.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strftime( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format* argument is changed from **const char *** to **const wchar_t ***.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Issue 6

The *wcsftime*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcslen — get wide-character string length

SYNOPSIS

```
#include <wchar.h>

size_t wcslen(const wchar_t *ws);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcslen*() function shall compute the number of wide-character codes in the wide-character string to which *ws* points, not including the terminating null wide-character code.

RETURN VALUE

The *wcslen*() function shall return the length of *ws*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcsncat — concatenate a wide-character string with part of another

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsncat*( ) function shall append not more than *n* wide-character codes (a null wide-character code and wide-character codes that follow it are not appended) from the array pointed to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null wide-character code shall always be appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wcsncat*( ) function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscat( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The *wcsncat*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcsncmp — compare part of two wide-character strings

SYNOPSIS

```
#include <wchar.h>

int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsncmp*( ) function shall compare not more than *n* wide-character codes (wide-character codes that follow a null wide-character code are not compared) from the array pointed to by *ws1* to the array pointed to by *ws2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

RETURN VALUE

Upon successful completion, *wcsncmp*( ) shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *ws2*, respectively.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscmp( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcsncpy — copy part of a wide-character string

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsncpy*( ) function shall copy not more than *n* wide-character codes (wide-character codes that follow a null wide-character code are not copied) from the array pointed to by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*, until *n* wide-character codes in all are written.

RETURN VALUE

The *wcsncpy*( ) function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

If there is no null wide-character code in the first *n* wide-character codes of the array pointed to by *ws2*, the result is not null-terminated.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscpy( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The *wcsncpy*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcspbrk — scan a wide-character string for a wide-character code

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcspbrk*( ) function shall locate the first occurrence in the wide-character string pointed to by *ws1* of any wide-character code from the wide-character string pointed to by *ws2*.

RETURN VALUE

Upon successful completion, *wcspbrk*( ) shall return a pointer to the wide-character code or a null pointer if no wide-character code from *ws2* occurs in *ws1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcschr( ), *wcsrchr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcsrchr — wide-character string scanning operation

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsrchr*( ) function shall locate the last occurrence of *wc* in the wide-character string pointed to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code shall be considered to be part of the wide-character string.

RETURN VALUE

Upon successful completion, *wcsrchr*( ) shall return a pointer to the wide-character code or a null pointer if *wc* does not occur in the wide-character string.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcschr( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcsrtombs — convert a wide-character string to a character string (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,
    size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsrtombs*( ) function shall convert a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, which shall also be stored. Conversion shall stop earlier in the following cases:

- When a code is reached that does not correspond to a valid character
- When the next character would exceed the limit of *len* total bytes to be stored in the array pointed to by *dst* (and *dst* is not a null pointer)

Each conversion shall take place as if by a call to the *wcrtomb*( ) function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcsrtombs*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *wcsrtombs*( ).

CX If the application uses any of the _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS functions, the application shall ensure that the *wcsrtombs*( ) function is called with a non-NULL *ps* argument.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

If conversion stops because a code is reached that does not correspond to a valid character, an encoding error occurs. In this case, the *wcsrtombs*( ) function shall store the value of the macro [EILSEQ] in *errno* and return (**size_t**)–1; the conversion state is undefined. Otherwise, it shall return the number of bytes in the resulting character sequence, not including the terminating null (if any).

ERRORS

The *wcsrtombs*( ) function may fail if:

CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

[EILSEQ] A wide-character code does not correspond to a valid character.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *wcrtomb*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

In the DESCRIPTION, a note on using this function in a threaded application is added.

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The *wcsrtombs*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcsspn — get the length of a wide substring

SYNOPSIS

```
#include <wchar.h>

size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsspn*( ) function shall compute the length (in wide characters) of the maximum initial segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character codes from the wide-character string pointed to by *ws2*.

RETURN VALUE

The *wcsspn*( ) function shall return the length of the initial substring of *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscspn( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The RETURN VALUE section is updated to indicate that *wcsspn*( ) returns the length of *ws1* rather that *ws1* itself.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

wcsstr — find a wide-character substring

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcsstr(const wchar_t *restrict ws1,
    const wchar_t *restrict ws2);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsstr*( ) function shall locate the first occurrence in the wide-character string pointed to by *ws1* of the sequence of wide characters (excluding the terminating null wide character) in the wide-character string pointed to by *ws2*.

RETURN VALUE

Upon successful completion, *wcsstr*( ) shall return a pointer to the located wide-character string, or a null pointer if the wide-character string is not found.

If *ws2* points to a wide-character string with zero length, the function shall return *ws1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcschr( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The *wcsstr*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

SYNOPSIS

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);
float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they shall decompose the input wide-character string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace*( ))
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character, then an optional exponent part
- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character, then an optional binary exponent part
- One of INF or INFINITY, or any other wide string equivalent except for case
- One of NAN or NAN(*n-wchar-sequence*$_{opt}$), or any other wide string ignoring case in the NAN part, where:

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of wide characters starting with the first digit or the radix character (whichever occurs first) shall be interpreted as a floating constant according to the rules of the C language, except that the radix character shall be used in place of a period, and that if neither an exponent part nor a radix character appears in a decimal floating-point number, or if a binary exponent part does not

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero shall be assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence shall be interpreted as negated. A wide-character sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it were a floating constant that is too large for the range of the return type. A wide-character sequence NAN or NAN(*n-wchar-sequence*$_{opt}$) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence part that does not have the expected form; the meaning of the *n*-wchar sequences is implementation-defined. A pointer to the final wide string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the conversion shall be rounded in an implementation-defined manner.

CX The radix character shall be as defined in the program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period (`'.'`).

CX In other than the C or POSIX locales, other implementation-defined subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

CX The *wcstod*( ) function shall not change the setting of *errno* if successful.

Since 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *wcstod*( ), *wcstof*( ), or *wcstold*( ), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value. If no conversion could be performed, 0 shall be returned CX and *errno* may be set to [EINVAL].

If the correct value is outside the range of representable values, ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to [ERANGE].

If the correct value would cause underflow, a value whose magnitude is no greater than the smallest normalized positive number in the return type shall be returned and *errno* set to [ERANGE].

ERRORS

The *wcstod*( ) function shall fail if:

[ERANGE] The value to be returned would cause overflow or underflow.

The *wcstod*( ) function may fail if:

CX [EINVAL] No conversion could be performed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in **<float.h>**) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the values of *L*, *D*, and *U* satisfy `"L <= D <= U"`. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswspace( ), *localeconv*( ), *scanf*( ), *setlocale*( ), *wcstol*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<float.h>**, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *wcstod*( ) prototype is updated.
- The *wcstof*( ) and *wcstold*( ) functions are added.
- If the correct value for *wcstod*( ) would cause underflow, the return value changed from 0 (as specified in Issue 5) to the smallest normalized positive number.
- The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively updated.

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second paragraph in the RETURN VALUE section.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

wcstoimax, wcstoumax — convert a wide-character string to an integer type

SYNOPSIS

```
#include <stddef.h>
#include <inttypes.h>

intmax_t wcstoimax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
uintmax_t wcstoumax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall be equivalent to the *wcstol*( ), *wcstoll*( ), *wcstoul*( ), and *wcstoull*( ) functions, respectively, except that the initial portion of the wide string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

If no conversion could be performed, zero shall be returned. If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to [ERANGE].

ERRORS

These functions shall fail if:

[EINVAL] The value of *base* is not supported.

[ERANGE] The value to be returned is not representable.

These functions may fail if:

[EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcstol( ), *wcstoul*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<inttypes.h>**, **<stddef.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcstok — split a wide-character string into tokens

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    wchar_t **restrict ptr);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

A sequence of calls to *wcstok*( ) shall break the wide-character string pointed to by *ws1* into a sequence of tokens, each of which shall be delimited by a wide-character code from the wide-character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar_t** pointer into which the *wcstok*( ) function shall store information necessary for it to continue scanning the same wide-character string.

The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by *ws2* may be different from call to call.

The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no such wide-character code is found, then there are no tokens in the wide-character string pointed to by *ws1* and *wcstok*( ) shall return a null pointer. If such a wide-character code is found, it shall be the start of the first token.

The *wcstok*( ) function shall then search from there for a wide-character code that *is* contained in the current separator string. If no such wide-character code is found, the current token extends to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a null wide character, which terminates the current token. The *wcstok*( ) function shall save a pointer to the following wide-character code, from which the next search for a token shall start.

Each subsequent call, with a null pointer as the value of the first argument, shall start searching from the saved pointer and behave as described above.

The implementation shall behave as if no function calls *wcstok*( ).

RETURN VALUE

Upon successful completion, the *wcstok*( ) function shall return a pointer to the first wide-character code of a token. Otherwise, if there is no token, *wcstok*( ) shall return a null pointer.

ERRORS

No errors are defined.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4.

Issue 5

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is added to the definition of *wcstok*( ) in the SYNOPSIS.

Issue 6

The *wcstok*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcstol, wcstoll — convert a wide-character string to a long integer

SYNOPSIS

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
    int base);
long long wcstoll(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **long**, **long long**, **unsigned long**, and **unsigned long long** representation, respectively. First, they shall decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace*( ))
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '−' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a '+' or '−' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code representations of 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first non-white-space wide-character code that is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide-character code, or if the first non-white-space wide-character code is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to the final wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

not a null pointer.

CX In other than the C or POSIX locales, other implementation-defined subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

CX These functions shall not change the setting of *errno* if successful.

Since 0, {LONG_MIN} or {LLONG_MIN} and {LONG_MAX} or {LLONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set *errno* to 0, then call *wcstol*( ) or *wcstoll*( ), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value, if any. If no CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX}, {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and *errno* set to [ERANGE].

ERRORS

These functions shall fail if:

CX [EINVAL] The value of *base* is not supported.

[ERANGE] The value to be returned is not representable.

These functions may fail if:

CX [EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalpha( ), *scanf*( ), *wcstod*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *wcstol*( ) prototype is updated.
- The *wcstoll*( ) function is added.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcstold — convert a wide-character string to a double-precision number

SYNOPSIS

```
#include <wchar.h>

long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
```

DESCRIPTION

Refer to *wcstod*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcstoll — convert a wide-character string to a long integer

SYNOPSIS

```
#include <wchar.h>

long long wcstoll(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

Refer to *wcstol*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcstombs — convert a wide-character string to a character string

SYNOPSIS

```
#include <stdlib.h>

size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcstombs*( ) function shall convert the sequence of wide-character codes that are in the array pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n* total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to *wctomb*( ), except that the shift state of *wctomb*( ) shall not be affected.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place CX between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs*( ) shall return the length required to convert the entire array regardless of the value of *n*, but no values are stored.

The *wcstombs*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

If a wide-character code is encountered that does not correspond to a valid character (of one or more bytes each), *wcstombs*( ) shall return (**size_t**)–1. Otherwise, *wcstombs*( ) shall return the number of bytes stored in the character array, not including any terminating null byte. The array shall not be null-terminated if the value returned is *n*.

ERRORS

The *wcstombs*( ) function may fail if:

CX [EILSEQ] A wide-character code does not correspond to a valid character.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen( ), *mbtowc*( ), *mbstowcs*( ), *wctomb*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

50228 **CHANGE HISTORY**

50229 First released in Issue 4. Derived from the ISO C standard.

50230 **Issue 6**

50231 The following new requirements on POSIX implementations derive from alignment with the
50232 Single UNIX Specification:

50233 • The DESCRIPTION states the effect of when *s* is a null pointer.

50234 • The [EILSEQ] error condition is added.

50235 The *wcstombs*( ) prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

wcstoul, wcstoull — convert a wide-character string to an unsigned long

SYNOPSIS

```
#include <wchar.h>

unsigned long wcstoul(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
unsigned long long wcstoull(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcstoul*( ) and *wcstoull*( ) functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively. First, they shall decompose the input wide-character string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace*( ))
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to an unsigned integer, and return the result.

If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a `'+'` or `'−'` sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix `'0'` optionally followed by a sequence of the digits `'0'` to `'7'` only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters `'a'` (or `'A'`) to `'f'` (or `'F'`) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a `'+'` or `'−'` sign, but not including an integer suffix. The letters from `'a'` (or `'A'`) to `'z'` (or `'Z'`) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first wide-character code that is not white space and is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide-character codes, or if the first wide-character code that is not white space is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to the final wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

not a null pointer.

CX In other than the C or POSIX locales, other implementation-defined subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

CX The *wcstoul*( ) function shall not change the setting of *errno* if successful.

Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and 0 is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *wcstoul*( ) or *wcstoull*( ), then check *errno*.

RETURN VALUE

Upon successful completion, the *wcstoul*( ) and *wcstoull*( ) functions shall return the converted CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If the correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX} respectively shall be returned and *errno* set to [ERANGE].

ERRORS

These functions shall fail if:

CX [EINVAL] The value of *base* is not supported.

[ERANGE] The value to be returned is not representable.

These functions may fail if:

CX [EINVAL] No conversion could be performed.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalpha( ), *scanf*( ), *wcstod*( ), *wcstol*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] error condition is added for when the value of *base* is not supported.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *wcstoul*() prototype is updated.
- The *wcstoull*() function is added.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

50328 **NAME**

50329 wcstoumax — convert a wide-character string to an integer type

50330 **SYNOPSIS**

50331 ```
#include <stddef.h>
```

50332 ```
#include <inttypes.h>
```

50333 ```
uintmax_t wcstoumax(const wchar_t *restrict nptr,
```

50334 ```
    wchar_t **restrict endptr, int base);
```

50335 **DESCRIPTION**

50336 Refer to *wcstoimax*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

NAME

wcswcs — find a wide substring (**LEGACY**)

SYNOPSIS

XSI
```
#include <wchar.h>

wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

The *wcswcs*( ) function shall locate the first occurrence in the wide-character string pointed to by *ws1* of the sequence of wide-character codes (excluding the terminating null wide-character code) in the wide-character string pointed to by *ws2*.

RETURN VALUE

Upon successful completion, *wcswcs*( ) shall return a pointer to the located wide-character string or a null pointer if the wide-character string is not found.

If *ws2* points to a wide-character string with zero length, the function shall return *ws1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function was not included in the final ISO/IEC 9899:1990/Amendment 1:1995 (E). Application developers are strongly encouraged to use the *wcsstr*( ) function instead.

RATIONALE

None.

FUTURE DIRECTIONS

This function may be withdrawn in a future version.

SEE ALSO

wcschr( ), *wcsstr*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 5

Marked EX.

Issue 6

This function is marked LEGACY.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wcswidth — number of column positions of a wide-character string

SYNOPSIS

XSI
```
#include <wchar.h>

int wcswidth(const wchar_t *pwcs, size_t n);
```

DESCRIPTION

The *wcswidth*( ) function shall determine the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

RETURN VALUE

The *wcswidth*( ) function either shall return 0 (if *pwcs* points to a null wide-character code), or return the number of column positions to be occupied by the wide-character string pointed to by *pwcs*, or return –1 (if any of the first *n* wide-character codes in the wide-character string pointed to by *pwcs* is not a printable wide-character code).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function was removed from the final ISO / IEC 9899: 1990 / Amendment 1: 1995 (E), and the return value for a non-printable wide character is not specified.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcwidth( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.103, Column Position, **<wchar.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U021 / 11 is applied. The function is marked as an extension.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcsxfrm — wide-character string transformation

SYNOPSIS

```
#include <wchar.h>

size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wcsxfrm*( ) function shall transform the wide-character string pointed to by *ws2* and place the resulting wide-character string into the array pointed to by *ws1*. The transformation shall be such that if *wcscmp*( ) is applied to two transformed wide strings, it shall return a value greater than, equal to, or less than 0, corresponding to the result of *wcscoll*( ) applied to the same two original wide-character strings. No more than *n* wide-character codes shall be placed into the resulting array pointed to by *ws1*, including the terminating null wide-character code. If *n* is 0, *ws1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

CX The *wcsxfrm*( ) function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *wcsxfrm*( ), then check *errno*.

RETURN VALUE

The *wcsxfrm*( ) function shall return the length of the transformed wide-character string (not including the terminating null wide-character code). If the value returned is *n* or more, the contents of the array pointed to by *ws1* are unspecified.

CX On error, the *wcsxfrm*( ) function may set *errno*, but no return value is reserved to indicate an error.

ERRORS

The *wcsxfrm*( ) function may fail if:

CX [EINVAL] The wide-character string pointed to by *ws2* contains wide-character codes outside the domain of the collating sequence.

EXAMPLES

None.

APPLICATION USAGE

The transformation function is such that two transformed wide-character strings can be ordered by *wcscmp*( ) as appropriate to collating sequence information in the program's locale (category *LC_COLLATE*).

The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of the *ws1* array prior to making the transformation.

RATIONALE

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

50444 **FUTURE DIRECTIONS**

50445 None.

50446 **SEE ALSO**

50447 *wcscmp*( ), *wcscoll*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

50448 **CHANGE HISTORY**

50449 First released in Issue 4. Derived from the MSE working draft.

50450 **Issue 5**

50451 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

50452 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

50453 **Issue 6**

50454 In previous versions, this function was required to return –1 on error.

50455 Extensions beyond the ISO C standard are marked.

50456 The following new requirements on POSIX implementations derive from alignment with the
50457 Single UNIX Specification:

50458 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
50459 added if no conversion could be performed.

50460 The *wcsxfrm*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wctob — wide-character to single-byte conversion

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int wctob(wint_t c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wctob*( ) function shall determine whether *c* corresponds to a member of the extended character set whose character representation is a single byte when in the initial shift state.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *wctob*( ) function shall return EOF if *c* does not correspond to a character with length one in the initial shift state. Otherwise, it shall return the single-byte representation of that character as an **unsigned char** converted to **int**.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

btowc( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wctomb — convert a wide-character code to a character

SYNOPSIS

```
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wctomb*( ) function shall determine the number of bytes needed to represent the character corresponding to the wide-character code whose value is *wchar* (including any change in the shift state). It shall store the character representation (possibly multiple bytes and any special bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most {MB_CUR_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state, and *wctomb*( ) shall be left in the initial shift state.

CX The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function shall be placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified.

The *wctomb*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *wctomb*( ).

RETURN VALUE

If *s* is a null pointer, *wctomb*( ) shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb*( ) shall return –1 if the value of *wchar* does not correspond to a valid character, or return the number of bytes that constitute the character corresponding to the value of *wchar*.

In no case shall the value returned be greater than the value of the {MB_CUR_MAX} macro.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

50536 **SEE ALSO**

50537 *mblen*( ), *mbtowc*( ), *mbstowcs*( ), *wcstombs*( ), the Base Definitions volume of IEEE Std 1003.1-2001,
50538 **<stdlib.h>**

50539 **CHANGE HISTORY**

50540 First released in Issue 4. Derived from the ANSI C standard.

50541 **Issue 6**

50542 Extensions beyond the ISO C standard are marked.

50543 In the DESCRIPTION, a note about reentrancy and thread-safety is added.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wctrans — define character mapping

SYNOPSIS

```
#include <wctype.h>

wctrans_t wctrans(const char *charclass);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wctrans*( ) function is defined for valid character mapping names identified in the current locale. The *charclass* is a string identifying a generic character mapping name for which codeset-specific information is required. The following character mapping names are defined in all locales: **tolower** and **toupper**.

The function shall return a value of type **wctrans_t**, which can be used as the second argument to subsequent calls of *towctrans*( ). The *wctrans*( ) function shall determine values of **wctrans_t** according to the rules of the coded character set defined by character mapping information in the program's locale (category *LC_CTYPE*). The values returned by *wctrans*( ) shall be valid until a call to *setlocale*( ) that modifies the category *LC_CTYPE*.

RETURN VALUE

CX The *wctrans*( ) function shall return 0 and may set *errno* to indicate the error if the given character mapping name is not valid for the current locale (category *LC_CTYPE*); otherwise, it shall return a non-zero object of type **wctrans_t** that can be used in calls to *towctrans*( ).

ERRORS

The *wctrans*( ) function may fail if:

CX [EINVAL] The character mapping name pointed to by *charclass* is not valid in the current locale.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

towctrans( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wctype.h>**

CHANGE HISTORY

First released in Issue 5. Derived from ISO / IEC 9899: 1990 / Amendment 1: 1995 (E).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wctype — define character class

SYNOPSIS

```
#include <wctype.h>

wctype_t wctype(const char *property);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wctype*( ) function is defined for valid character class names as defined in the current locale. The *property* argument is a string identifying a generic character class for which codeset-specific type information is required. The following character class names shall be defined in all locales:

alnum	**digit**	**punct**
alpha	**graph**	**space**
blank	**lower**	**upper**
cntrl	**print**	**xdigit**

Additional character class names defined in the locale definition file (category *LC_CTYPE*) can also be specified.

The function shall return a value of type **wctype_t**, which can be used as the second argument to subsequent calls of *iswctype*( ). The *wctype*( ) function shall determine values of **wctype_t** according to the rules of the coded character set defined by character type information in the program's locale (category *LC_CTYPE*). The values returned by *wctype*( ) shall be valid until a call to *setlocale*( ) that modifies the category *LC_CTYPE*.

RETURN VALUE

The *wctype*( ) function shall return 0 if the given character class name is not valid for the current locale (category *LC_CTYPE*); otherwise, it shall return an object of type **wctype_t** that can be used in calls to *iswctype*( ).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswctype( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wctype.h>**

CHANGE HISTORY

First released in Issue 4.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Issue 5

The following change has been made in this issue for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wcwidth — number of column positions of a wide-character code

SYNOPSIS

XSI
```
#include <wchar.h>

int wcwidth(wchar_t wc);
```

DESCRIPTION

The *wcwidth*( ) function shall determine the number of column positions required for the wide character *wc*. The application shall ensure that the value of *wc* is a character representable as a **wchar_t**, and is a wide-character code corresponding to a valid character in the current locale.

RETURN VALUE

The *wcwidth*( ) function shall either return 0 (if *wc* is a null wide-character code), or return the number of column positions to be occupied by the wide-character code *wc*, or return –1 (if *wc* does not correspond to a printable wide-character code).

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcswidth( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 6

The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wmemchr — find a wide character in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wmemchr*( ) function shall locate the first occurrence of *wc* in the initial *n* wide characters of the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if no valid occurrence of *wc* is found.

RETURN VALUE

The *wmemchr*( ) function shall return a pointer to the located wide character, or a null pointer if the wide character does not occur in the object.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemcmp( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wmemcmp — compare wide characters in memory

SYNOPSIS

```
#include <wchar.h>

int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wmemcmp*( ) function shall compare the first *n* wide characters of the object pointed to by *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall behave as if the two objects compare equal.

RETURN VALUE

The *wmemcmp*( ) function shall return an integer greater than, equal to, or less than zero, respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object pointed to by *ws2*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wmemcpy — copy wide characters in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wmemcpy*( ) function shall copy *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall copy zero wide characters.

RETURN VALUE

The *wmemcpy*( ) function shall return the value of *ws1*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemmove*( ), *wmemset*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The *wmemcpy*( ) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wmemmove — copy wide characters in memory with overlapping areas

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wmemmove*( ) function shall copy *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary array are copied into the object pointed to by *ws1*.

This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall copy zero wide characters.

RETURN VALUE

The *wmemmove*( ) function shall return the value of *ws1*.

ERRORS

No errors are defined

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemcpy*( ), *wmemset*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

wmemset — set wide characters in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

The *wmemset*( ) function shall copy the value of *wc* into each of the first *n* wide characters of the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy zero wide characters.

RETURN VALUE

The *wmemset*( ) functions shall return the value of *ws*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemcpy*( ), *wmemmove*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

wordexp, wordfree — perform word expansions

SYNOPSIS

```
#include <wordexp.h>

int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
    int flags);
void wordfree(wordexp_t *pwordexp);
```

DESCRIPTION

The *wordexp*( ) function shall perform word expansions as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6, Word Expansions, subject to quoting as in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.2, Quoting, and place the list of expanded words into the structure pointed to by *pwordexp*.

The *words* argument is a pointer to a string containing one or more words to be expanded. The expansions shall be the same as would be performed by the command line interpreter if *words* were the part of a command line representing the arguments to a utility. Therefore, the application shall ensure that *words* does not contain an unquoted <newline> or any of the unquoted shell special characters `'|'`, `'&'`, `';'`, `'<'`, `'>'` except in the context of command substitution as specified in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.3, Command Substitution. It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. The application shall ensure that every member of *words* which it expects to have expanded by *wordexp*( ) does not contain an unquoted initial comment character. The application shall also ensure that any words which it intends to be ignored (because they begin or continue a comment) are deleted from *words*. If the argument *words* contains an unquoted comment character (number sign) that is the beginning of a token, *wordexp*( ) shall either treat the comment character as a regular character, or interpret it as a comment indicator and ignore the remainder of *words*.

The structure type **wordexp_t** is defined in the **<wordexp.h>** header and includes at least the following members:

Member Type	Member Name	Description
size_t	*we_wordc*	Count of words matched by *words*.
**char ** **	*we_wordv*	Pointer to list of expanded words.
size_t	*we_offs*	Slots to reserve at the beginning of *pwordexp->we_wordv*.

The *wordexp*( ) function shall store the number of generated words into *pwordexp->we_wordc* and a pointer to a list of pointers to words in *pwordexp->we_wordv*. Each individual field created during field splitting (see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.5, Field Splitting) or pathname expansion (see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.6, Pathname Expansion) shall be a separate word in the *pwordexp->we_wordv* list. The words shall be in order as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6, Word Expansions. The first pointer after the last word pointer shall be a null pointer. The expansion of special parameters described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.5.2, Special Parameters is unspecified.

It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp*( ) function shall allocate other space as needed, including memory pointed to by *pwordexp->we_wordv*. The *wordfree*( ) function frees any memory associated with *pwordexp* from a previous call to *wordexp*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The *flags* argument is used to control the behavior of *wordexp*(). The value of *flags* is the bitwise-inclusive OR of zero or more of the following constants, which are defined in **<wordexp.h>**:

WRDE_APPEND — Append words generated to the ones from a previous call to *wordexp*().

WRDE_DOOFFS — Make use of *pwordexp->we_offs*. If this flag is set, *pwordexp->we_offs* is used to specify how many null pointers to add to the beginning of *pwordexp->we_wordv*. In other words, *pwordexp->we_wordv* shall point to *pwordexp->we_offs* null pointers, followed by *pwordexp->we_wordc* word pointers, followed by a null pointer.

WRDE_NOCMD — If the implementation supports the utilities defined in the Shell and Utilities volume of IEEE Std 1003.1-2001, fail if command substitution, as specified in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.3, Command Substitution, is requested.

WRDE_REUSE — The *pwordexp* argument was passed to a previous successful call to *wordexp*(), and has not been passed to *wordfree*(). The result shall be the same as if the application had called *wordfree*() and then called *wordexp*() without WRDE_REUSE.

WRDE_SHOWERR — Do not redirect *stderr* to **/dev/null**.

WRDE_UNDEF — Report error on an attempt to expand an undefined shell variable.

The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to *wordexp*(). The following rules apply to applications when two or more calls to *wordexp*() are made with the same value of *pwordexp* and without intervening calls to *wordfree*():

1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.
2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.
3. After the second and each subsequent call, *pwordexp->we_wordv* shall point to a list containing the following:
   a. Zero or more null pointers, as specified by WRDE_DOOFFS and *pwordexp->we_offs*
   b. Pointers to the words that were in the *pwordexp->we_wordv* list before the call, in the same order as before
   c. Pointers to the new words generated by the latest call, in the specified order
4. The count returned in *pwordexp->we_wordc* shall be the total number of words from all of the calls.
5. The application can change any of the fields after a call to *wordexp*(), but if it does it shall reset them to the original value before a subsequent call, using the same *pwordexp* value, to *wordfree*() or *wordexp*() with the WRDE_APPEND or WRDE_REUSE flag.

If the implementation supports the utilities defined in the Shell and Utilities volume of IEEE Std 1003.1-2001, and *words* contains an unquoted character—<newline>, `'|'`, `'&'`, `';'`, `'<'`, `'>'`, `'('`, `')'`, `'{'`, `'}'`—in an inappropriate context, *wordexp*() shall fail, and the number of expanded words shall be 0.

Unless WRDE_SHOWERR is set in *flags*, *wordexp*() shall redirect *stderr* to **/dev/null** for any utilities executed as a result of command substitution while expanding *words*. If WRDE_SHOWERR is set, *wordexp*() may write messages to *stderr* if syntax errors are detected while expanding *words*.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The application shall ensure that if WRDE_DOOFFS is set, then *pwordexp->we_offs* has the same value for each *wordexp*( ) call and *wordfree*( ) call using a given *pwordexp*.

The following constants are defined as error return values:

WRDE_BADCHAR	One of the unquoted characters—<newline>, '|', '&', ';', '<', '>', '(', ')', '{', '}'—appears in *words* in an inappropriate context.
WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.
WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in *flags*.
WRDE_NOSPACE	Attempt to allocate memory failed.
WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.

RETURN VALUE

Upon successful completion, *wordexp*( ) shall return 0. Otherwise, a non-zero value, as described in **<wordexp.h>**, shall be returned to indicate an error. If *wordexp*( ) returns the value WRDE_NOSPACE, then *pwordexp->we_wordc* and *pwordexp->we_wordv* shall be updated to reflect any words that were successfully expanded. In other cases, they shall not be modified.

The *wordfree*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *wordexp*( ) function is intended to be used by an application that wants to do all of the shell's expansions on a word or words obtained from a user. For example, if the application prompts for a filename (or list of filenames) and then uses *wordexp*( ) to process the input, the user could respond with anything that would be valid as input to the shell.

The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to prevent a user from executing shell commands. Disallowing unquoted shell special characters also prevents unwanted side effects, such as executing a command or writing a file.

RATIONALE

This function was included as an alternative to *glob*( ). There had been continuing controversy over exactly what features should be included in *glob*( ). It is hoped that by providing *wordexp*( ) (which provides all of the shell word expansions, but which may be slow to execute) and *glob*( ) (which is faster, but which only performs pathname expansion, without tilde or parameter expansion) this will satisfy the majority of applications.

While *wordexp*( ) could be implemented entirely as a library routine, it is expected that most implementations run a shell in a subprocess to do the expansion.

Two different approaches have been proposed for how the required information might be presented to the shell and the results returned. They are presented here as examples.

One proposal is to extend the *echo* utility by adding a **−q** option. This option would cause *echo* to add a backslash before each backslash and <blank> that occurs within an argument. The *wordexp*( ) function could then invoke the shell as follows:

```
(void) strcpy(buffer, "echo -q");
(void) strcat(buffer, words);
if ((flags & WRDE_SHOWERR) == 0)
```


--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
        (void) strcat(buffer, "2>/dev/null");
    f = popen(buffer, "r");
```

The *wordexp*( ) function would read the resulting output, remove unquoted backslashes, and break into words at unquoted <blank>s. If the WRDE_NOCMD flag was set, *wordexp*( ) would have to scan *words* before starting the subshell to make sure that there would be no command substitution. In any case, it would have to scan *words* for unquoted special characters.

Another proposal is to add the following options to *sh*:

–w *wordlist*
This option provides a wordlist expansion service to applications. The words in *wordlist* shall be expanded and the following written to standard output:

1. The count of the number of words after expansion, in decimal, followed by a null byte
2. The number of bytes needed to represent the expanded words (not including null separators), in decimal, followed by a null byte
3. The expanded words, each terminated by a null byte

If an error is encountered during word expansion, *sh* exits with a non-zero status after writing the former to report any words successfully expanded

–P Run in ''protected'' mode. If specified with the **–w** option, no command substitution shall be performed.

With these options, *wordexp*( ) could be implemented fairly simply by creating a subprocess using *fork*( ) and executing *sh* using the line:

```
execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

after directing standard error to **/dev/null**.

It seemed objectionable for a library routine to write messages to standard error, unless explicitly requested, so *wordexp*( ) is required to redirect standard error to **/dev/null** to ensure that no messages are generated, even for commands executed for command substitution. The WRDE_SHOWERR flag can be specified to request that error messages be written.

The WRDE_REUSE flag allows the implementation to avoid the expense of freeing and reallocating memory, if that is possible. A minimal implementation can call *wordfree*( ) when WRDE_REUSE is set.

FUTURE DIRECTIONS

None.

SEE ALSO

fnmatch( ), *glob*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wordexp.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, Chapter 2, Shell Command Language

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

The DESCRIPTION is updated to avoid use of the term ''must'' for application requirements.

The **restrict** keyword is added to the *wordexp*( ) prototype for alignment with the ISO / IEC 9899: 1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

51020 **NAME**

51021 wprintf — print formatted wide-character output

51022 **SYNOPSIS**

51023 `#include <stdio.h>`

51024 `#include <wchar.h>`

51025 `int wprintf(const wchar_t *restrict format, ...);`

51026 **DESCRIPTION**

51027 Refer to *fwprintf*( ).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

pwrite, write — write on a file

SYNOPSIS

```
#include <unistd.h>

ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
    off_t offset);
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

DESCRIPTION

The *write*( ) function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the open file descriptor, *fildes*.

Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the *write*( ) function may detect and return errors as described below. In the absence of errors, or if error detection is not performed, the *write*( ) function shall return zero and have no other results. If *nbyte* is zero and the file is not a regular file, the results are unspecified.

On a regular file or other file capable of seeking, the actual writing of data shall proceed from the position in the file indicated by the file offset associated with *fildes*. Before successful return from *write*( ), the file offset shall be incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file shall be set to this file offset.

On a file not capable of seeking, writing shall always take place starting at the current position. The value of a file offset associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file prior to each write and no intervening file modification operation shall occur between changing the file offset and the write operation.

XSI If a *write*( ) requests that more bytes be written than there is room for (for example, the process' file size limit or the physical end of a medium), only as many bytes as there is room for shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would give a failure return (except as noted below).

XSI If the request would cause the file size to exceed the soft file size limit for the process and there is no room for any bytes to be written, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

If *write*( ) is interrupted by a signal before it writes any data, it shall return –1 with *errno* set to [EINTR].

If *write*( ) is interrupted by a signal after it successfully writes some data, it shall return the number of bytes written.

If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

After a *write*( ) to a regular file has successfully returned:

- Any successful *read*( ) from each byte position in the file that was modified by that write shall return the data specified by the *write*( ) for that position until such byte positions are again modified.
- Any subsequent successful *write*( ) to the same byte position in the file shall overwrite that file data.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request shall append to the end of the pipe.
- Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the O_NONBLOCK flag of the file status flags is set.
- If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on normal completion it shall return *nbyte*.
- If the O_NONBLOCK flag is set, *write*( ) requests shall be handled differently, in the following ways:
  - The *write*( ) function shall not block the thread.
  - A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there is sufficient space available in the pipe, *write*( ) shall transfer all the data and return the number of bytes requested. Otherwise, *write*( ) shall transfer no data and return –1 with *errno* set to [EAGAIN].
  - A write request for more than {PIPE_BUF} bytes shall cause one of the following:
    - When at least one byte can be written, transfer what it can and return the number of bytes written. When all data previously written to the pipe is read, it shall transfer at least {PIPE_BUF} bytes.
    - When no data can be written, transfer no data, and return –1 with *errno* set to [EAGAIN].

When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-blocking writes and cannot accept the data immediately:

- If the O_NONBLOCK flag is clear, *write*( ) shall block the calling thread until the data can be accepted.
- If the O_NONBLOCK flag is set, *write*( ) shall not block the thread. If some data can be written without blocking the thread, *write*( ) shall write what it can and return the number of bytes written. Otherwise, it shall return –1 and set *errno* to [EAGAIN].

Upon successful completion, where *nbyte* is greater than 0, *write*( ) shall mark for update the *st_ctime* and *st_mtime* fields of the file, and if the file is a regular file, the S_ISUID and S_ISGID bits of the file mode may be cleared.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *fildes*.

If *fildes* refers to a socket, *write*( ) shall be equivalent to *send*( ) with no flags set.

SIO If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.

If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

SHM If *fildes* refers to a shared memory object, the result of the *write*( ) function is unspecified.

TYM If *fildes* refers to a typed memory object, the result of the *write*( ) function is unspecified.

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

XSR If *fildes* refers to a STREAM, the operation of *write*( ) shall be determined by the values of the minimum and maximum *nbyte* range (packet size) accepted by the STREAM. These values are determined by the topmost STREAM module. If *nbyte* falls within the packet size range, *nbyte* bytes shall be written. If *nbyte* does not fall within the range and the minimum packet size value is 0, *write*( ) shall break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, *write*( ) shall fail with *errno* set to [ERANGE]. Writing a zero-length buffer (*nbyte* is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue I_SWROPT *ioctl*( ) to enable zero-length messages to be sent across the pipe or FIFO.

When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:

- If O_NONBLOCK is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), *write*( ) shall block until data can be accepted.
- If O_NONBLOCK is set and the STREAM cannot accept data, *write*( ) shall return –1 and set *errno* to [EAGAIN].
- If O_NONBLOCK is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, *write*( ) shall terminate and return the number of bytes written.

In addition, *write*( ) shall fail if the STREAM head has processed an asynchronous error before the call. In this case, the value of *errno* does not reflect the result of *write*( ), but reflects the prior error.

XSI The *pwrite*( ) function shall be equivalent to *write*( ), except that it writes into a given position without changing the file pointer. The first three arguments to *pwrite*( ) are the same as *write*( ) with the addition of a fourth argument offset for the desired position inside the file.

RETURN VALUE

XSI Upon successful completion, *write*( ) and *pwrite*( ) shall return the number of bytes actually written to the file associated with *fildes*. This number shall never be greater than *nbyte*. Otherwise, –1 shall be returned and *errno* set to indicate the error.

ERRORS

XSI The *write*( ) and *pwrite*( ) functions shall fail if:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the thread would be delayed in the *write*( ) operation.

[EBADF] The *fildes* argument is not a valid file descriptor open for writing.

[EFBIG] XSI An attempt was made to write a file that exceeds the implementation-defined maximum file size or the process' file size limit, and there was no room for any bytes to be written.

[EFBIG] The file is a regular file, *nbyte* is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with *fildes*.

[EINTR] The write operation was terminated due to the receipt of a signal, and no data was transferred.

[EIO] The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

[ENOSPC] There was no free space remaining on the device containing the file.

[EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.

XSR [ERANGE] The transfer request size was outside the range supported by the STREAMS file associated with *fildes*.

The *write*( ) function shall fail if:

[EAGAIN] or [EWOULDBLOCK]
The file descriptor is for a socket, is marked O_NONBLOCK, and write would block.

[ECONNRESET] A write was attempted on a socket that is not connected.

[EPIPE] A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling process.

XSI The *write*( ) and *pwrite*( ) functions may fail if:

XSR [EINVAL] The STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

[EIO] A physical I/O error has occurred.

[ENOBUFS] Insufficient resources were available in the system to perform the operation.

[ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

XSR [ENXIO] A hangup occurred on the STREAM being written to.

XSR A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, *errno* is set to the value included in the error message.

The *write*( ) function may fail if:

[EACCES] A write was attempted on a socket and the calling process does not have appropriate privileges.

[ENETDOWN] A write was attempted on a socket and the local network interface used to reach the destination is down.

[ENETUNREACH]
A write was attempted on a socket and no route to the network is present.

XSI The *pwrite*( ) function shall fail and the file pointer remain unchanged if:

XSI [EINVAL] The *offset* argument is invalid. The value is negative.

XSI [ESPIPE] *fildes* is associated with a pipe or FIFO.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

Writing from a Buffer

The following example writes data from the buffer pointed to by *buf* to the file associated with the file descriptor *fd*.

```
#include <sys/types.h>
#include <string.h>
...
char buf[20];
size_t nbytes;
ssize_t bytes_written;
int fd;
...
strcpy(buf, "This is a test\n");
nbytes = strlen(buf);

bytes_written = write(fd, buf, nbytes);
...
```

APPLICATION USAGE

None.

RATIONALE

See also the RATIONALE section in *read*( ).

An attempt to write to a pipe or FIFO has several major characteristics:

- *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not interleaved with data from any other process. This is useful when there are multiple writers sending data to a single reader. Applications need to know how large a write request can be expected to be performed atomically. This maximum is called {PIPE_BUF}. This volume of IEEE Std 1003.1-2001 does not say whether write requests for more than {PIPE_BUF} bytes are atomic, but requires that writes of {PIPE_BUF} or fewer bytes shall be atomic.
- *Blocking/immediate*: Blocking is only possible with O_NONBLOCK clear. If there is enough space for all the data requested to be written immediately, the implementation should do so. Otherwise, the process may block; that is, pause until enough space is available for writing. The effective size of a pipe or FIFO (the maximum amount that can be written in one operation without blocking) may vary dynamically, depending on the implementation, so it is not possible to specify a fixed value for it.
- *Complete/partial/deferred*: A write request:

```
int fildes;
size_t nbyte;
ssize_t ret;
char *buf;

ret = write(fildes, buf, nbyte);
```

may return:

Complete *ret=nbyte*

Partial *ret<nbyte*

This shall never happen if *nbyte*≤{PIPE_BUF}. If it does happen (with *nbyte*>{PIPE_BUF}), this volume of IEEE Std 1003.1-2001 does not guarantee

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

atomicity, even if *ret*≤{PIPE_BUF}, because atomicity is guaranteed according to the amount *requested*, not the amount *written*.

Deferred: *ret*=−1, *errno*=[EAGAIN]

This error indicates that a later request may succeed. It does not indicate that it *shall* succeed, even if *nbyte*≤{PIPE_BUF}, because if no process reads from the pipe or FIFO, the write never succeeds. An application could usefully count the number of times [EAGAIN] is caused by a particular value of *nbyte*>{PIPE_BUF} and perhaps do later writes with a smaller value, on the assumption that the effective size of the pipe may have decreased.

Partial and deferred writes are only possible with O_NONBLOCK set.

The relations of these properties are shown in the following tables:

Write to a Pipe or FIFO with O_NONBLOCK *clear*			
Immediately Writable:	**None**	**Some**	*nbyte*
nbyte≤{PIPE_BUF}	Atomic blocking *nbyte*	Atomic blocking *nbyte*	Atomic immediate *nbyte*
nbyte>{PIPE_BUF}	Blocking *nbyte*	Blocking *nbyte*	Blocking *nbyte*

If the O_NONBLOCK flag is clear, a write request shall block if the amount writable immediately is less than that requested. If the flag is set (by *fcntl*( )), a write request shall never block.

Write to a Pipe or FIFO with O_NONBLOCK *set*			
Immediately Writable:	**None**	**Some**	*nbyte*
nbyte≤{PIPE_BUF}	−1, [EAGAIN]	−1, [EAGAIN]	Atomic *nbyte*
nbyte>{PIPE_BUF}	−1, [EAGAIN]	<*nbyte* or −1, [EAGAIN]	≤*nbyte* or −1, [EAGAIN]

There is no exception regarding partial writes when O_NONBLOCK is set. With the exception of writing to an empty pipe, this volume of IEEE Std 1003.1-2001 does not specify exactly when a partial write is performed since that would require specifying internal details of the implementation. Every application should be prepared to handle partial writes when O_NONBLOCK is set and the requested amount is greater than {PIPE_BUF}, just as every application should be prepared to handle partial writes on other kinds of file descriptors.

The intent of forcing writing at least one byte if any can be written is to assure that each write makes progress if there is any room in the pipe. If the pipe is empty, {PIPE_BUF} bytes must be written; if not, at least some progress must have been made.

Where this volume of IEEE Std 1003.1-2001 requires −1 to be returned and *errno* set to [EAGAIN], most historical implementations return zero (with the O_NDELAY flag set, which is the historical predecessor of O_NONBLOCK, but is not itself in this volume of IEEE Std 1003.1-2001). The error indications in this volume of IEEE Std 1003.1-2001 were chosen so that an application can distinguish these cases from end-of-file. While *write*( ) cannot receive an indication of end-of-file, *read*( ) can, and the two functions have similar return values. Also, some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file indication; for those systems, a return value of zero from *write*( ) indicates a successful write of an end-of-file indication.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Implementations are allowed, but not required, to perform error checking for *write*( ) requests of zero bytes.

The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be written to a pipe in a single operation) was considered, but rejected, because this concept would unnecessarily limit application writing.

See also the discussion of O_NONBLOCK in *read*( ).

Writes can be serialized with respect to other reads and writes. If a *read*( ) of file data can be proven (by any means) to occur after a *write*( ) of the data, it must reflect that *write*( ), even if the calls are made by different processes. A similar requirement applies to multiple write operations to the same file position. This is needed to guarantee the propagation of data from *write*( ) calls to subsequent *read*( ) calls. This requirement is particularly significant for networked file systems, where some caching schemes violate these semantics.

Note that this is specified in terms of *read*( ) and *write*( ). The XSI extensions *readv*( ) and *writev*( ) also obey these semantics. A new ''high-performance'' write analog that did not follow these serialization requirements would also be permitted by this wording. This volume of IEEE Std 1003.1-2001 is also silent about any effects of application-level caching (such as that done by *stdio*).

This volume of IEEE Std 1003.1-2001 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

This volume of IEEE Std 1003.1-2001 does not specify behavior of concurrent writes to a file from multiple processes. Applications should use some form of concurrency control.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *creat*( ), *dup*( ), *fcntl*( ), *getrlimit*( ), *lseek*( ), *open*( ), *pipe*( ), *ulimit*( ), *writev*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<stropts.h>**, **<sys/uio.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

The *pwrite*( ) function is added.

Issue 6

The DESCRIPTION states that the *write*( ) function does not block the thread. Previously this said ''process'' rather than ''thread''.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The DESCRIPTION now states that if *write*( ) is interrupted by a signal after it has successfully written some data, it returns the number of bytes written. In the POSIX.1-1988 standard, it was optional whether *write*( ) returned the number of bytes written, or whether it returned –1 with *errno* set to [EINTR]. This is a FIPS requirement.
- The following changes are made to support large files:
  - For regular files, no data transfer occurs past the offset maximum established in the open file description associated with the *fildes*.
  - A second [EFBIG] error condition is added.
- The [EIO] error condition is added.
- The [EPIPE] error condition is added for when a pipe has only one end open.
- The [ENXIO] optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *write*( ) results are unspecified for typed memory objects.

The following error conditions are added for operations on sockets: [EAGAIN], [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].

The [EIO] error is changed to ‘‘may fail’’.

The [ENOBUFS] error is added for sockets.

The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN], and [ENETUNREACH].

The *writev*( ) function is split out into a separate reference page.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NAME

writev — write a vector

SYNOPSIS

XSI
```
#include <sys/uio.h>

ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);
```

DESCRIPTION

The *writev*( ) function shall be equivalent to *write*( ), except as described below. The *writev*( ) function shall gather output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*–1]. The *iovcnt* argument is valid if greater than 0 and less than or equal to {IOV_MAX}, as defined in **<limits.h>**.

Each *iovec* entry specifies the base address and length of an area in memory from which data should be written. The *writev*( ) function shall always write a complete area before proceeding to the next.

If *fildes* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0, *writev*( ) shall return 0 and have no other effect. For other file types, the behavior is unspecified.

If the sum of the *iov_len* values is greater than {SSIZE_MAX}, the operation shall fail and no data shall be transferred.

RETURN VALUE

Upon successful completion, *writev*( ) shall return the number of bytes actually written. Otherwise, it shall return a value of –1, the file-pointer shall remain unchanged, and *errno* shall be set to indicate an error.

ERRORS

Refer to *write*( ).

In addition, the *writev*( ) function shall fail if:

[EINVAL] The sum of the *iov_len* values in the *iov* array would overflow an **ssize_t**.

The *writev*( ) function may fail and set *errno* to:

[EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

EXAMPLES

Writing Data from an Array

The following example writes data from the buffers specified by members of the *iov* array to the file associated with the file descriptor *fd*.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
...
ssize_t bytes_written;
int fd;
char *buf0 = "short string\n";
char *buf1 = "This is a longer string\n";
char *buf2 = "This is the longest string in this example\n";
int iovcnt;
struct iovec iov[3];
```


System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        iov[0].iov_base = buf0;
        iov[0].iov_len = strlen(buf0);
        iov[1].iov_base = buf1;
        iov[1].iov_len = strlen(buf1);
        iov[2].iov_base = buf2;
        iov[2].iov_len = strlen(buf2);
        ...
        iovcnt = sizeof(iov) / sizeof(struct iovec);

        bytes_written = writev(fd, iov, iovcnt);
        ...
```

APPLICATION USAGE

None.

RATIONALE

Refer to *write*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

readv( ), *write*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<sys/uio.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

Split out from the *write*( ) reference page.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

51413 **NAME**

51414 wscanf — convert formatted wide-character input

51415 **SYNOPSIS**

51416 `#include <stdio.h>`

51417 `#include <wchar.h>`

51418 `int wscanf(const wchar_t *restrict format, ... );`

51419 **DESCRIPTION**

51420 Refer to *fwscanf*( ).

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NAME

y0, y1, yn — Bessel functions of the second kind

SYNOPSIS

XSI
```
#include <math.h>

double y0(double x);
double y1(double x);
double yn(int n, double x);
```

DESCRIPTION

The *y0*(), *y1*(), and *yn*() functions shall compute Bessel functions of *x* of the second kind of orders 0, 1, and *n*, respectively.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the relevant Bessel value of *x* of the second kind.

If *x* is NaN, NaN shall be returned.

If the *x* argument to these functions is negative, −HUGE_VAL or NaN shall be returned, and a domain error may occur.

If *x* is 0.0, −HUGE_VAL shall be returned and a range error may occur.

If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

If the correct result would cause overflow, −HUGE_VAL or 0.0 shall be returned and a range error may occur.

ERRORS

These functions may fail if:

Domain Error	The value of *x* is negative.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.
Range Error	The value of *x* is 0.0, or the correct result would cause overflow.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.
Range Error	The value of *x* is too large in magnitude, or the correct result would cause underflow.
	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *j0*( ), the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The DESCRIPTION is updated to avoid use of the term ‘‘must’’ for application requirements.

The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling with the ISO/IEC 9899:1999 standard.

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Index

_CS_PATH 214
_CS_XBS5_ILP32_OFF32_CFLAGS....214
_CS_XBS5_ILP32_OFF32_LDFLAGS....214
_CS_XBS5_ILP32_OFF32_LIBS....214
_CS_XBS5_ILP32_OFFBIG_CFLAGS....214
_CS_XBS5_ILP32_OFFBIG_LDFLAGS....214
_CS_XBS5_ILP32_OFFBIG_LIBS....214
_CS_XBS5_LP64_OFF64_CFLAGS....214
_CS_XBS5_LP64_OFF64_LDFLAGS....214
_CS_XBS5_LP64_OFF64_LIBS....214
_CS_XBS5_LPBIG_OFFBIG_CFLAGS....214
_CS_XBS5_LPBIG_OFFBIG_LDFLAGS....214
_CS_XBS5_LPBIG_OFFBIG_LIBS....214
_exit....85, 307, 1592
_Exit( )....307, **85**
FILE....128
_IOFBF....1285, 1327
_IOLBF....376, 1327
_IONBF....1285, 1327
LINE....128
_longjmp( )....**86**
_LVL....17
_MAX....16
_MIN....16
_PC constants
used in pathconf....399
_PC_ALLOC_SIZE_MIN....399
_PC_ASYNC_IO....399
_PC_CHOWN_RESTRICTED....399
_PC_FILESIZEBITS....399
_PC_LINK_MAX....399
_PC_MAX_CANON....399
_PC_MAX_INPUT....399
_PC_NAME_MAX....399
_PC_NO_TRUNC....399
_PC_PATH_MAX....399
_PC_PIPE_BUF....399
_PC_PRIO_IO....399
_PC_REC_INCR_XFER_SIZE....399
_PC_REC_MAX_XFER_SIZE....399
_PC_REC_MIN_XFER_SIZE....399
_PC_REC_XFER_ALIGN....399
_PC_SYMLINK_MAX....399
_PC_SYNC_IO....399
_PC_VDISABLE....399
_POSIX2 constants
in sysconf....1473
_POSIX2_CHAR_TERM....1475
_POSIX2_C_BIND....1475
_POSIX2_C_DEV....1475
_POSIX2_C_VERSION....1475
_POSIX2_FORT_DEV....1475
_POSIX2_FORT_RUN....1475
_POSIX2_LOCALEDEF....1475
_POSIX2_PBS....1475
_POSIX2_PBS_ACCOUNTING....1475
_POSIX2_PBS_LOCATE....1475
_POSIX2_PBS_MESSAGE....1475
_POSIX2_PBS_TRACK....1475
_POSIX2_SW_DEV....1475
_POSIX2_UPE....1475
_POSIX2_VERSION....1475
POSIX....15, 17
_POSIX_ADVISORY_INFO....1473
_POSIX_ASYNCHRONOUS_IO....1473
_POSIX_ASYNC_IO....399
_POSIX_BARRIERS....1473
_POSIX_CHOWN_RESTRICTED....190, 399, 401
_POSIX_CLOCK_SELECTION....1473
_POSIX_CPUTIME....1473
_POSIX_C_SOURCE....14
_POSIX_FILE_LOCKING....1474
_POSIX_FSYNC....1474
_POSIX_JOB_CONTROL....1474
_POSIX_MAPPED_FILES....1474
_POSIX_MEMLOCK....1474
_POSIX_MEMLOCK_RANGE....1474
_POSIX_MEMORY_PROTECTION....1474
_POSIX_MESSAGE_PASSING....1474
_POSIX_MONOTONIC_CLOCK....1474
_POSIX_MULTI_PROCESS....1474
_POSIX_NO_TRUNC....399
_POSIX_OPEN_MAX....556
_POSIX_PRIORITIZED_IO....42, 1474
_POSIX_PRIORITY_SCHEDULING....42, 1474
_POSIX_PRIO_IO....399
_POSIX_READER_WRITER_LOCKS....1474
_POSIX_REALTIME_SIGNALS....1474
_POSIX_REGEXP....1474
_POSIX_SAVED_IDS....1474
_POSIX_SEMAPHORES....1474

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

_POSIX_SHARED_MEMORY_OBJECTS 1474
_POSIX_SHELL 1474
_POSIX_SOURCE.....14
_POSIX_SPAWN 1474
_POSIX_SPIN_LOCKS.....1474
_POSIX_SPORADIC_SERVER.....1474
_POSIX_SYNCHRONIZED_IO 1474
_POSIX_SYNC_IO 399
_POSIX_THREADS.....241, 1474, 1533
_POSIX_THREAD_ATTR_STACKADDR.....1474
_POSIX_THREAD_ATTR_STACKSIZE 1474
_POSIX_THREAD_CPUTIME.....1474
_POSIX_THREAD_PRIORITY_SCHEDULING ....
.....1474
_POSIX_THREAD_PRIO_INHERIT.....1474
_POSIX_THREAD_PRIO_PROTECT 1474
_POSIX_THREAD_PROCESS_SHARED.....1092
.....1474
_POSIX_THREAD_SAFE_FUNCTIONS.....241
.....1474, 1533
_POSIX_THREAD_SPORADIC_SERVER 1474
_POSIX_TIMEOUTS.....1474
_POSIX_TIMERS 1474
_POSIX_TRACE.....1474
_POSIX_TRACE_EVENT_FILTER 1474
_POSIX_TRACE_EVENT_NAME_MAX..939, 941
_POSIX_TRACE_INHERIT.....1474
_POSIX_TRACE_LOG 1474
_POSIX_TRACE_SYS_MAX 936
_POSIX_TRACE_USER_EVENT_MAX.....941
_POSIX_TYPED_MEMORY_OBJECTS 1474
_POSIX_V6_ILP32_OFF32.....1474
_POSIX_V6_ILP32_OFFBIG.....1474
_POSIX_V6_LP64_OFF64.....1474
_POSIX_V6_LPBIG_OFFBIG 1475
_POSIX_VDISABLE 399
_POSIX_VERSION.....1474, 1557
_PROCESS.....17
_PTHREAD_THREADS_MAX.....1066
_REGEX_VERSION 1475
_SC constants
in sysconf 1473
_SC_2_CHAR_TERM 1475
_SC_2_C_BIND.....1475
_SC_2_C_DEV.....1475
_SC_2_C_VERSION.....1475
_SC_2_FORT_DEV.....1475
_SC_2_FORT_RUN 1475
_SC_2_LOCALEDEF 1475
_SC_2_PBS_ACCOUNTING 1475
_SC_2_PBS_LOCATE 1475
_SC_2_PBS_MESSAGE 1475
_SC_2_PBS_TRACK.....1475
_SC_2_SW_DEV 1475
_SC_2_UPE 1475
_SC_2_VERSION 863, 1475
_SC_ADVISORY_INFO 1473
_SC_AIO_LISTIO_MAX 1473
_SC_AIO_MAX.....1473
_SC_AIO_PRIO_DELTA_MAX 1473
_SC_ARG_MAX 1473
_SC_ASYNCHRONOUS_IO 1473
_SC_ATEXIT_MAX 1473
_SC_BARRIERS 1473
_SC_BC_BASE_MAX.....1473
_SC_BC_DIM_MAX.....1473
_SC_BC_SCALE_MAX.....1473
_SC_BC_STRING_MAX 1473
_SC_CHILD_MAX 1473
_SC_CLK_TCK 1473, 1529
_SC_CLOCK_SELECTION 1473
_SC_COLL_WEIGHTS_MAX.....1473
_SC_CPUTIME.....1473
_SC_DELAYTIMER_MAX.....1473
_SC_EXPR_NEST_MAX 1473
_SC_FILE_LOCKING 1474
_SC_FSYNC 1474
_SC_GETGR_R_SIZE_MAX.....507, 510, 1473
_SC_GETPW_R_SIZE_MAX 548, 551, 1473
_SC_IOV_MAX.....1473
_SC_JOB_CONTROL 1474
_SC_LINE_MAX.....1473
_SC_LOGIN_NAME_MAX.....1473
_SC_MEMLOCK.....1474
_SC_MEMLOCK_RANGE 1474
_SC_MEMORY_PROTECTION 1474
_SC_MESSAGE_PASSING.....1474
_SC_MONOTONIC_CLOCK.....1474
_SC_MQ_OPEN_MAX.....1473
_SC_MQ_PRIO_MAX 1473
_SC_MULTI_PROCESS.....1474
_SC_NGROUPS_MAX.....1473
_SC_OPEN_MAX.....1473
_SC_PAGESIZE.....776, 868, 1475
_SC_PAGE_SIZE 776, 1475
_SC_PRIORITIZED_IO 1474
_SC_PRIORITY_SCHEDULING.....1474
_SC_READER_WRITER_LOCKS 1474
_SC_REALTIME_SIGNALS 1474
_SC_REGEXP 1474
_SC_REGEX_VERSION 1475
_SC_RE_DUP_MAX 1475

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

_SC_RTSIG_MAX 1475
_SC_SAVED_IDS 1474
_SC_SEMAPHORES 1474
_SC_SEM_NSEMS_MAX 1475
_SC_SEM_VALUE_MAX 1475
_SC_SHARED_MEMORY_OBJECTS 1474
_SC_SHELL 1474
_SC_SIGQUEUE_MAX 1475
_SC_SPAWN 1474
_SC_SPIN_LOCKS 1474
_SC_SPORADIC_SERVER 1474
_SC_STREAM_MAX 1475
_SC_SYMLOOP_MAX 1475
_SC_SYNCHRONIZED_IO 1474
_SC_THREADS 1474
_SC_THREAD_ATTR_STACKADDR 1474
_SC_THREAD_ATTR_STACKSIZE 1474
_SC_THREAD_CPUTIME 1474
_SC_THREAD_DESTRUCTOR_ITERATIONS 1475
_SC_THREAD_KEYS_MAX 1475
_SC_THREAD_PRIORITY_SCHEDULING 1474
_SC_THREAD_PRIO_INHERIT 1474
_SC_THREAD_PRIO_PROTECT 1474
_SC_THREAD_PROCESS_SHARED 1474
_SC_THREAD_SAFE_FUNCTIONS 1474
_SC_THREAD_SPORADIC_SERVER 1474
_SC_THREAD_STACK_MIN 1475
_SC_THREAD_THREADS_MAX 1475
_SC_TIMEOUTS 1474
_SC_TIMERS 1474
_SC_TIMER_MAX 1475
_SC_TRACE 1474
_SC_TRACE_EVENT_FILTER 1474
_SC_TRACE_INHERIT 1474
_SC_TRACE_LOG 1474
_SC_TTY_NAME_MAX 1475
_SC_TYPED_MEMORY_OBJECTS 1474
_SC_TZNAME_MAX 1475
_SC_V6_ILP32_OFF32 1474
_SC_V6_ILP32_OFFBIG 1474
_SC_V6_LP64_OFF64 1474
_SC_V6_LPBIG_OFFBIG 1475
_SC_VERSION 1474
_SC_XBS5_ILP32_OFF32 1475
_SC_XBS5_ILP32_OFFBIG 1475
_SC_XBS5_LP64_OFF64 1475
_SC_XBS5_LPBIG_OFFBIG 1475
_SC_XOPEN_CRYPT 1475
_SC_XOPEN_ENH_I18N 1475
_SC_XOPEN_LEGACY 1475
_SC_XOPEN_REALTIME 1475
_SC_XOPEN_REALTIME_THREADS 1475
_SC_XOPEN_SHM 1475
_SC_XOPEN_UNIX 1475
_SC_XOPEN_VERSION 1475
_SC_XOPEN_XCU_VERSION 1476
_setjmp 86
_t 17
_TIME 17
_tolower( ) **88**
_toupper( ) **89**
_XBS5_ILP32_OFF32 1475
_XBS5_ILP32_OFFBIG 1475
_XBS5_LP64_OFF64 1475
_XBS5_LPBIG_OFFBIG 1475
_XOPEN_CRYPT 1475
_XOPEN_ENH_I18N 1475
_XOPEN_LEGACY 1475
_XOPEN_REALTIME 337, 1475
_XOPEN_REALTIME_THREADS 1475
_XOPEN_SHM 1475
_XOPEN_SOURCE 14
_XOPEN_UNIX 1475
_XOPEN_VERSION 1475
_XOPEN_XCU_VERSION 1476
a64l( ) **90**
ABDAY_1 834
abort( ) **92**
abs( ) **94**
accept( ) **95**
access( ) **97**
acos( ) **100**
acosf 100
acosh( ) **102**
acoshf 102
acoshl 102
acosl( ) 100, **104**
ACTION 584
address information 426
address string 426
addrinfo structure 426
ADV **3**
ADVANCED REALTIME ....196, 200, 800-801, 864 ....866, 868, 870, 872, 875, 883, 886 ....888-890, 892, 894, 896, 898, 900 ....902, 904-911, 964, 966, 1043, 1088, 1259
ADVANCED REALTIME THREADS ....1012, 1014 ....1016, 1018, 1020-1021, 1060, 1141 ....1143, 1145
AF_ 18
AIO **3**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

AIO_ 16
aio_ 16
AIO_ALLDONE 105
aio_cancel( ) **105**
AIO_CANCELED 105
aio_error( ) **107**
aio_fsync( ) **108**
AIO_LISTIO_MAX 687, 1473
AIO_MAX 687, 1473
AIO_NOTCANCELED 105
AIO_PRIO_DELTA_MAX 42, 1473
aio_read( ) **110**
aio_return( ) **113**
aio_suspend( ) **114**
aio_write( ) **116**
AI_ALL 427
AI_CANONNAME 427
AI_INET6 427
AI_NUMERICHOST 427
AI_NUMERICSERV 427
AI_PASSIVE 427
AI_V4MAPPED 427
alarm( ) **119**
anycast 67
ANYMARK 620
appropriate privileges 98, 400
argc 302
ARG_MAX 22, 297, 300, 304, 1473
asctime( ) **121**
asctime_r 121
asin( ) **123**
asinf 123
asinh( ) **125**
asinhf 125
asinhl 125
asinl( ) 123, **127**
assert( ) **128**
async-signal-safe 979
atan( ) **129**
atan2( ) **131**
atan2f 131
atan2l 131
atanf( ) 129, **133**
atanh( ) **134**
atanhf 134
atanhl 134
atanl( ) 129, **136**
atexit( ) **137**
ATEXIT_MAX 137, 1473
atof( ) **138**
atoi( ) **139**
atol( ) **141**
atoll 141
attributes, clock-resolution 76, 914
attributes, creation-time 76, 914
attributes, generation-version 75, 914
attributes, inheritance 76, 916
attributes, log-full-policy 74, 76, 916, 919
attributes, log-max-size 76, 917, 919
attributes, max-data-size 76, 919-920
attributes, stream-full-policy 72-73, 76, 917
attributes, stream-min-size 76, 920
attributes, trace-name 76, 914
attributes, truncation-status 939
background 1305
background process 1504
BAR **3**
basename( ) **142**
baud rate functions 180
bcmp( ) **144**
bcopy( ) **145**
BC_ constants
in sysconf 1473
BC_BASE_MAX 1473
BC_DIM_MAX 1473
BC_SCALE_MAX 1473
BC_STRING_MAX 1473
BE **4**
bind( ) **146**
bi_ 16
BOOT_TIME 284-285
broadcasting a condition 1030
BSD 119, 191, 310, 334, 343, 401
....539, 669, 756, 823, 1176, 1216
....1224, 1305, 1348, 1372, 1496, 1519
....1557, 1592
bsd_signal( ) **148**
bsearch( ) **150**
btowc( ) **153**
buffer cache 452
BUFSIZ 1285
BUS_ 18
byte-oriented stream 36
byte-stream mode 1173
bzero( ) **154**
cabs( ) **155**
cabsf 155
cabsl 155
cacos( ) **156**
cacosf 156
cacosh( ) **157**
cacoshf 157

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

cacoshl....157
cacosl( )....156, **158**
calloc( )....**159**
can....1
cancel-safe....1133
cancelability state....1067, 1133
cancelability states....54
cancelability type....1067, 1133
canceling execution of a thread....1022
canonical name....427
carg( )....**161**
cargf....161
cargl....161
casin( )....**162**
casinf....162
casinh( )....**163**
casinhf....163
casinhl....163
casinl( )....162, **164**
catan( )....**165**
catanf....165
catanh( )....**166**
catanhf....166
catanhl....166
catanl( )....165, **167**
catclose( )....**168**
catgets( )....**169**
catopen( )....**171**
CBAUD....19
cbrt( )....**173**
cbrtf....173
cbrtl....173
ccos( )....**174**
ccosf....174
ccosh( )....**175**
ccoshf....175
ccoshl....175
ccosl( )....174, **176**
CD....**4**
ceil( )....**177**
ceilf....177
ceill....177
cexp( )....**179**
cexpf....179
cexpl....179
cfgetispeed( )....**180**
cfgetospeed( )....**182**
cfsetispeed( )....**183**
cfsetospeed( )....**184**
change current working directory....186
change file modes....189
change owner and group of file....191
CHAR_MAX....697, 699
chdir( )....**185**
CHILD_MAX....395, 1473
chmod( )....**187**
chown( )....**190**
cimag( )....**193**
cimagf....193
cimagl....193
CLD_....18
clearerr( )....**194**
clock tick....119, 1477, 1529
clock ticks/second....1473
clock( )....**195**
clock-resolution attribute....76, 914
CLOCKS_PER_SEC....195
CLOCK_....17
clock_....17
clock_getcpuclockid( )....**196**
clock_getres( )....**197**
clock_gettime....197
CLOCK_MONOTONIC....49, 201
clock_nanosleep( )....**200**
CLOCK_PROCESS_CPUTIME_ID....49
CLOCK_REALTIME....49, 197, 201
....823, 1088, 1259, 1521
clock_settime( )....197, **203**
CLOCK_THREAD_CPUTIME_ID....50
clog( )....**204**
clogf....204
clogl....204
close a file....207
close( )....**205**
closedir( )....**208**
closelog( )....**210**
cmsg_....16
CMSG_....18
COLL_WEIGHTS_MAX....1473
command interpreter
portable....1592
compare thread IDs....1055
compilation environment....**13**
condition variable initialization attributes....1041
conforming application....1389
conforming application, strictly....119, 302
confstr( )....**214**
conj( )....**217**
conjf....217
conjl....217
connect( )....**218**
control data....38

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

control-normal **1173**
conversion descriptor 297, 302, 590-593
conversion specification 404, 435, 468
..... 477, 1424, 1428, 1441
modified 1430
conversion specifier
modified 1442
copysign( ) **221**
copysignf 221
copysignl 221
core 1593
core file 309
cos( ) **222**
cosf 222
cosh( ) **224**
coshf 224
coshl 224
cosl( ) 222, **226**
covert channel 669
cpow( ) **227**
cpowf 227
cpowl 227
cproj( ) **228**
cprojf 228
cprojl 228
CPT **4**
creal( ) **229**
crealf 229
creall 229
creat( ) **230**
create a per-process timer 1522
create an interprocess channel 856
create session and set process group ID 1317
creation-time attribute 76, 914
CRYPT 232, 270, 1298
crypt( ) **232**
CS **4**
csin( ) **234**
csinf 234
csinh( ) **235**
csinhf 235
csinhl 235
csinl( ) 234, **236**
csqrt( ) **237**
csqrtf 237
csqrtl 237
ctan( ) **238**
ctanf 238
ctanh( ) **239**
ctanhf 239
ctanhl 239
ctanl( ) 238, **240**
ctermid( ) **241**
ctime( ) **243**
ctime_r 243
CX **4**
c_ 17
data key creation 1071
data messages 38
data type **80**
DATEMSK 495
daylight **245**, 1549
DBL_MANT_DIG 177, 377
DBL_MAX_EXP 177, 377
DBM 246-247
dbm_ 16
DBM_ 18
dbm_clearerr( ) **246**
dbm_close 246
dbm_delete 246
dbm_error 246
dbm_fetch 246
dbm_firstkey 246
DBM_INSERT 246
dbm_nextkey 246
dbm_open 246
DBM_REPLACE 246
dbm_store 246
DEAD_PROCESS 284-285
DEFECHO 19
deferred cancelability 1067
delay process execution 1388
DELAYTIMER_MAX 1473, 1526
dependency ordering 259
descriptive name 426
destroying a mutex 1077
destroying condition variables 1034
destructor functions 1071
detaching a thread 1053
difftime( ) **250**
DIR 80, 208, 845, 1179, 1181, 1219, 1245, 1512
directive 404, 435, 468, 477, 1441
directory operations 846
dirent structure 846
dirname( ) **251**
div( ) **253**
dlclose( ) **254**
dlerror( ) **256**
dlopen( ) **258**
dlsym( ) **261**
dot 846, 1216
dot-dot 846, 1216

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

drand48( ) **263**
dup( ) **265**
dup2 265
dynamic package initialization 1109
d_ 16
E2BIG **22**
EACCES **22**
EADDRINUSE **22**
EADDRNOTAVAIL **22**
EAFNOSUPPORT **22**
EAGAIN **22**, 28
EALREADY **22**
EBADF **22**
EBADMSG **22**
EBUSY **23**
ECANCELED **23**
ECHILD **23**
ECHOCTL 19
ECHOKE 19
ECHOPRT 19
ECONNABORTED **23**
ECONNREFUSED **23**
ECONNRESET **23**
ecvt( ) **268**
EDEADLK **23**
EDESTADDRREQ **23**
EDOM **23**
EDQUOT **23**
EEXIST **23**
EFAULT **23**
EFBIG **23**
effective group ID 191, 304, 513
effective user ID 98, 304, 669
EHOSTUNREACH 23
EIDRM **23**
Eighth Edition UNIX 1661
EILSEQ **24**, 37
EINPROGRESS **24**, 42
EINTR **24**, 57, 933
EINVAL **24**, 920, 933
EIO **24**
EISCONN **24**
EISDIR **24**
ELOOP **24**
ELSIZE 726
EMFILE **24**
EMLINK **24**
EMPTY 285
EMSGSIZE **24**
EMULTIHOP **24**
ENAMETOOLONG **25**
encrypt( ) **270**
endgrent( ) **272**
endhostent( ) **274**
endnetent( ) **276**
endprotoent( ) **278**
endpwent( ) **280**
endservent( ) **282**
endutxent( ) **284**
ENETDOWN **25**
ENETRESET **25**
ENETUNREACH **25**
ENFILE **25**
ENOBUFS **25**
ENODATA **25**
ENODEV **25**
ENOENT **25**
ENOEXEC **25**
ENOLCK **25**
ENOLINK **25**
ENOMEM **25**
ENOMSG **25**
ENOPROTOOPT **25**
ENOSPC **26**
ENOSR **26**
ENOSTR **26**
ENOSYS **26**
ENOTCONN **26**
ENOTDIR **26**
ENOTEMPTY **26**
ENOTSOCK **26**
ENOTSUP **26**
ENOTTY **26**
ENTRY 584
environ **287**, 303
envp 303
ENXIO **26**
EOPNOTSUPP **26**
EOVERFLOW **26**
EPERM **26**
EPIPE **26**
EPROTO **27**
EPROTONOSUPPORT **27**
EPROTOTYPE **27**
erand48( ) 263, **288**
ERANGE **27**
erf( ) **289**
erfc( ) **291**
erfcf 291
erfcl 291
erff( ) 289, **293**
erfl 289, 293

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EROFS **27**
errno **294**
error descriptions 483
error numbers 21
  additional 28
ESPIPE **27**
ESRCH **27**
EST5EDT 1549
ESTALE **27**
ETIME **27**
ETIMEDOUT **27**
ETXTBSY **27**
EWOULDBLOCK **27**
examine and change blocked signals 1140
examine and change signal action 1348
EXDEV **28**
exec **296**
  of shell scripts 303
exec family ... 98, 207, 333, 374, 397, 979, 1306, 1592
execl 296
execle 296
execlp 296
execute a file 302
execution time monitoring 49
execv 296
execve 296
execvp 296
exit( ) **307**
EXIT_FAILURE 307
EXIT_SUCCESS 307, 310
exp( ) **312**
exp2( ) **314**
exp2f 314
exp2l 314
expf 312
expl 312
expm1( ) **316**
expm1f 316
expm1l 316
EXPR_NEST_MAX 1473
EXTA 19
EXTB 19
extension
  CX 4
  OH 7
  XSI 11
extensions to setlocale 1300
fabs( ) **318**
fabsf 318
fabsl 318
fattach( ) **319**
fchdir( ) **322**
fchmod( ) **323**
fchown( ) **325**
fclose( ) **327**
fcntl( ) **329**
fcvt( ) 268, **336**
FD **4**
fdatasync( ) **337**
fdetach( ) **338**
fdim( ) **340**
fdimf 340
fdiml 340
fdopen( ) **342**
fds_ 16
FD_ 16, 18
fd_ 16
FD_CLOEXEC 35, 171, 297, 329, 593
  ..... 838, 846, 856, 876, 883, 1329
FD_CLR( ) **84**, 974
FD_ISSET 84, 974
FD_SET 84, 974
FD_ZERO 84, 974
feature test macro 13, 488
  _POSIX_C_SOURCE 14
  _XOPEN_SOURCE 14
feclearexcept( ) **344**
fegetenv( ) **345**
fegetexceptflag( ) **346**
fegetround( ) **347**
feholdexcept( ) **349**
feof( ) **350**
feraiseexcept( ) **351**
ferror( ) **352**
fesetenv( ) 345, **353**
fesetexceptflag( ) 346, **354**
fesetround( ) 347, **355**
fetestexcept( ) **356**
feupdateenv( ) **358**
fflush( ) **360**
ffs( ) **363**
fgetc( ) **364**
fgetpos( ) **366**
fgets( ) **368**
fgetwc( ) **370**
fgetws( ) **372**
FIFO 758-759, 842, 1660
FILE 80, 194, 327, 342, 350, 352
  ..... 360, 364, 366, 368, 370, 372
  ..... 374-375, 390, 404, 416, 418, 420
  ..... 422-423, 430, 435, 442, 445, 454
  ..... 466-468, 475, 477, 486-487, 573

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

.....852, 862, 1148-1149, 1161, 1218
.....1285, 1327, 1409, 1531, 1559-1560
.....1577, 1579, 1581, 1583, 1585, 1587
file
locking 332
file accessibility 98
file control 332
FILE object.....34
file permission bits 98
file permissions.....98, 401, 1406
file position indicator.....34
fileno( ) **374**
FILESIZEBITS.....399
FIND.....584
find string token.....1454
flockfile( ).....**375**
floor( ).....**377**
floorf.....377
floorl 377
FLT_RADIX.....716
FLT_ROUNDS.....379
FLUSH.....18
FLUSHO 19
FLUSHR.....614
FLUSHRW 614
FLUSHW 614
fma( ) **379**
fmaf.....379
fmal.....379
fmax( ) **381**
fmaxf 381
fmaxl.....381
fmin( ).....**382**
fminf 382
fminl 382
FMNAMESZ 613
fmod( ).....**383**
fmodf.....383
fmodl.....383
fmtmsg( ) **385**
fnmatch( ) **388**
FNM_ 18
FNM_NOESCAPE.....388
FNM_NOMATCH.....388
FNM_PATHNAME.....388
FNM_PERIOD.....388
fopen( ).....**390**
FOPEN_MAX.....342, 391, 862, 1531
foreground 1305
fork handler 980
fork( ).....**394**
forkall 397
format of entries.....**11**
fpathconf( ).....**399**
fpclassify( ) **403**
FPE_.....18
fprintf( ).....**404**
fputc( ).....**416**
fputs( ).....**418**
fputwc( ) **420**
fputws( ).....**422**
FQDN 528
FR **4**
fread( ).....**423**
free( ).....**425**
freeaddrinfo( ).....**426**
freopen( ) **430**
frexp( ).....**433**
frexpf 433
frexpl 433
FSC **5**
fscanf( ).....**435**
fseek( ) **442**
fseeko 442
fsetpos( ) **445**
fstat( ).....**447**
fstatvfs( ).....**449**
fsync( ).....**452**
ftell( ) **454**
ftello.....454
ftime( ).....**456**
ftok( ) **458**
ftruncate( ).....**460**
ftrylockfile( ) 375, **462**
FTW 18, 829-830
ftw( ) **463**
FTW_CHDIR 829
FTW_D 463, 829
FTW_DEPTH.....829
FTW_DNR 463, 829-830
FTW_DP 829
FTW_F 463, 829
FTW_MOUNT 829
FTW_NS 463, 829-830
FTW_PHYS.....829
FTW_SL.....463, 829
FTW_SLN.....829
fully-qualified domain name 528
functions.....**13**
implementation.....13
use.....13
funlockfile( ).....375, **466**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fwide( ) **467**
fwprintf( ) **468**
fwrite( ) **475**
fwscanf( ) **477**
f_ 16
F_ 18
F_DUPFD 265, 329, 331-332
F_GETFD 329, 331-332
F_GETFL 329, 331-332
F_GETLK 330-332
F_GETOWN 329, 331
F_LOCK 705
F_RDLCK 332
F_SETFD 329, 331-333
F_SETFL 329, 331-332
F_SETLK 330-332
F_SETLKW 55, 330-332
F_SETOWN 329, 332
F_TEST 705
F_TLOCK 705
F_ULOCK 705
F_UNLCK 330-331
F_WRLCK 332
gai_strerror( ) **483**
gcvt( ) 268, **484**
generation-version attribute 75, 914
get configurable pathname variables 401
get configurable system variables 1476
get file status 1406
get process times 1529
get supplementary group IDs 512
get system time 1519
get thread ID 1131
get user name 522
getaddrinfo( ) 426, **485**
GETALL 1266
getc( ) **486**
getchar( ) **489**
getchar_unlocked( ) 487, **490**
getcontext( ) **491**
getcwd( ) **493**
getc_unlocked( ) **487**
getdate( ) **495**
getdate_err 495
getegid( ) **500**
getenv( ) 303, **501**
geteuid( ) **504**
getgid( ) **505**
getgrent( ) 272, **506**
getgrgid( ) **507**
getgrgid_r 507
getgrnam( ) **510**
getgrnam_r 510
getgroups( ) **512**
gethostbyaddr( ) **514**
gethostbyname 514
gethostent( ) 274, **516**
gethostid( ) **517**
gethostname( ) **518**
getitimer( ) **519**
getlogin( ) **521**
getlogin_r 521
getmsg( ) **524**
getnameinfo( ) **528**
GETNCNT 1266-1267
getnetbyaddr( ) 276, **531**
getnetbyname 276, 531
getnetent 276, 531
getopt( ) **532**
getpeername( ) **537**
getpgid( ) **538**
getpgrp( ) **539**
GETPID 1266-1267
getpid( ) **540**
getpmsg( ) 524, **541**
getppid( ) **542**
getpriority( ) **543**
getprotent 546
getprotobyname( ) 278, **546**
getprotobynumber 278, 546
getprotoent 278
getpwent( ) 280, **547**
getpwnam( ) **548**
getpwnam_r 548
getpwuid( ) **551**
getpwuid_r 551
getrlimit( ) **554**
getrusage( ) **557**
gets( ) **559**
getservbyname( ) 282, **560**
getservbyport 282, 560
getservent 282, 560
getsid( ) **561**
getsockname( ) **562**
getsockopt( ) **563**
getsubopt( ) **566**
gettimeofday( ) **570**
getuid( ) **571**
getutxent 284
getutxent( ) **572**
getutxid 284, 572
getutxline 284, 572

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

GETVAL....1266-1267
getwc( )....**573**
getwchar( )....**574**
getwd( )....494, **575**
GETZCNT....1266-1267
glob( )....**576**
globfree....576
GLOB_....18
GLOB_ constants
  error returns of glob....578
  used in glob....576
GLOB_ABORTED....578
GLOB_APPEND....576-577
GLOB_DOOFFS....576-577
GLOB_ERR....576, 578
GLOB_MARK....577
GLOB_NOCHECK....577-578
GLOB_NOESCAPE....577
GLOB_NOMATCH....578
GLOB_NOSORT....577
GLOB_NOSPACE....578
gl_....16
GMT0....1549
gmtime( )....**580**
gmtime_r....580
grantpt( )....**582**
granularity of clock....456
HALT....386
hcreate( )....**584**
hdestroy....584
high resolution sleep....823
host name....426
htonl( )....**587**
htons....587
HUGE_VAL....134, 177, 224, 312, 314
  ....316, 340, 377, 588, 677, 681, 708
  ....712, 714, 825, 827, 969, 1221, 1226
  ....1230, 1385, 1449, 1485, 1516, 1619
HUGE_VALF....377, 1221, 1449
HUGE_VALL....377, 1221, 1449
hypot( )....**588**
hypotf....588
hypotl....588
h_....16
h_errno....**583**
iconv( )....**590**
iconv_close( )....**592**
iconv_open( )....**593**
ic_....16
IEEE Std 754-1985....3
IEEE Std 854-1987....3
ifc_....16
ifra_....16
ifru_....16
if_....16
IF_....18
if_freenameindex( )....**595**
if_indextoname( )....**596**
if_nameindex( )....**597**
if_nametoindex( )....**598**
ILL_....18
ilogb( )....**599**
ilogbf....599
ilogbl....599
imaxabs( )....**601**
imaxdiv( )....**602**
implementation-defined....1
IMPLINK_....18
in6_....16
IN6_....18
INADDR_....18
index( )....**603**
inet_....16
inet_addr( )....**604**
inet_ntoa....604
inet_ntop( )....**606**
inet_pton....606
Inf....123
INF....407, 471
INFINITY....407, 471
INFO....386
infu_....16
inheritance attribute....76, 916
init....310, 669
initialize a named semaphore....1255
initialize an unnamed semaphore....1252
initializing a mutex....1077
initializing condition variables....1034
initstate( )....**608**
INIT_PROCESS....284-285
input and output rationale....1175
insque( )....**610**
international environment....1300
Internet Protocols....66
INT_MAX....599
INT_MIN....94
in_....16
IN_....18
ioctl( )....**613**
iov_....17
IOV_....18
IOV_MAX....1186, 1473, 1664

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IP6 ..... **5**
IPC..... **39**, 806, 808, 811, 813, 1271, 1275, 1339, 1341
ipc_..... 16
IPC_ 18
IPC_ constants
  used in semctl 1266
  used in shmctl 1337
IPC_CREAT 807, 1269, 1340
IPC_EXCL 807, 1269
IPC_NOWAIT 809-810, 812-813, 1272
IPC_PRIVATE 807, 1269, 1340
IPC_RMID 805, 1267, 1337
IPC_SET 805, 1267, 1337
IPC_STAT 805, 1266, 1337
IPPORT_ 18
IPPROTO_ 18
IPv4 67
IPv4-compatible address 68
IPv4-mapped address 68
IPv6 67
  compatibility with IPv4 68
  interface identification 68
  options 69
IPv6 address
  anycast 67
  loopback 68
  multicast 67
  unicast 67
  unspecified 68
IPV6_ 18
IPV6_JOIN_GROUP 69
IPV6_LEAVE_GROUP 69
IPV6_MULTICAST_HOPS 69
IPV6_MULTICAST_IF 69
IPV6_MULTICAST_LOOP 69
IPV6_UNICAST_HOPS 69
IPV6_V6ONLY 69
ip_ 16
IP_ 18
isalnum( ) **625**
isalpha( ) **626**
isascii( ) **627**
isastream( ) **628**
isatty( ) **629**
isblank( ) **630**
iscntrl( ) **631**
isdigit( ) **632**
isfinite( ) **633**
isgraph( ) **634**
isgreater( ) **635**
isgreaterequal( ) **636**
isinf( ) **637**
isless( ) **638**
islessequal( ) **639**
islessgreater( ) **640**
islower( ) **641**
isnan( ) **643**
isnormal( ) **644**
ISO C standard 3, 119, 302, 332
  ..... 488, 728, 1169, 1216, 1300, 1348
  ..... 1372, 1519
isprint( ) **645**
ispunct( ) **646**
isspace( ) **647**
isunordered( ) **648**
isupper( ) **649**
iswalnum( ) **650**
iswalpha( ) **651**
iswblank( ) **652**
iswcntrl( ) **653**
iswctype( ) **654**
iswdigit( ) **656**
iswgraph( ) **657**
iswlower( ) **658**
iswprint( ) **659**
iswpunct( ) **660**
iswspace( ) **661**
iswupper( ) **662**
iswxdigit( ) **663**
isxdigit( ) **664**
ITIMER_PROF 519
ITIMER_REAL 519
ITIMER_VIRTUAL 519
it_ 16-17
I_ 18
I_ATMARK 619-620
I_CANPUT 620
I_CKBAND 620
I_FDINSERT 616
I_FIND 615
I_FLUSH 613
I_FLUSHBAND 614
I_GETBAND 620
I_GETCLTIME 620
I_GETSIG 615
I_GRDOPT 616, 1173
I_GWROPT 618
I_LINK 621
I_LIST 619
I_LOOK 613
I_NREAD 616
I_PEEK 615

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

I_PLINK....622
I_POP....613
I_PUNLINK....622
I_PUSH....613
I_RECVFD....22, 619
I_SENDFD....618-619
I_SETCLTIME....205, 620
I_SETSIG....614-615
I_SRDOPT....615-616, 1173
I_STR....617
I_SWROPT....618, 1658
I_UNLINK....621
j0( )....**665**
j1....665
jn....665
job control....310, 539, 669, 1305, 1317, 1476, 1592
jrand48( )....263, **667**
JST-9....1549
kill( )....**668**
killpg( )....**671**
l64a( )....90, **672**
labs( )....**673**
LANG....171
last close....1333
LASTMARK....620
lchown( )....**674**
lcong48( )....263, **676**
LC_ALL....297, 699, 834, 1299, 1301
LC_COLLATE....576-577, 1299-1300, 1416
....1464, 1603, 1637
LC_CTYPE....153, 654, 736, 738
....740, 742-743, 745, 747, 1299-1300
....1536-1540, 1598, 1614, 1629
....1639-1640, 1642-1643
LC_MESSAGES....171, 1299-1300, 1422
LC_MONETARY....699, 1299-1300, 1425
LC_NUMERIC....268, 405, 435, 468, 477
....699, 1299-1300, 1425, 1449, 1619
LC_TIME....496, 834, 1299-1300
ldexp( )....**677**
ldexpf....677
ldexpl....677
ldiv( )....**679**
legacy....1
lfind( )....**680**, 726
lgamma( )....**681**
lgammaf....681
lgammal....681
LIFO....57
LINE_MAX....1473
link to a file....685
link( )....**683**
LINK_MAX....24, 399, 683, 1215
LIO_....16
lio_....16
lio_listio( )....**686**
LIO_NOP....686
LIO_NOWAIT....686
LIO_READ....686
LIO_WAIT....686
LIO_WRITE....686
list directed I/O....688
listen( )....**689**
llabs( )....673, **691**
lldiv( )....679, **692**
LLONG_MAX....1457, 1625
LLONG_MIN....1457
llrint( )....**693**
llrintf....693
llrintl....693
llround( )....**695**
llroundf....695
llroundl....695
load ordering....259
LOBLK....19
localeconv( )....**697**
localtime( )....**702**
localtime_r....702
lockf( )....**705**
locking....332
advisory....333
mandatory....333
locking and unlocking a mutex....1085
log( )....**708**
log-full-policy attribute....74, 76, 916, 919, 937
log-max-size attribute....76, 917, 919
log10( )....**710**
log10f....710
log10l....710
log1p( )....**712**
log1pf....712
log1pl....712
log2( )....**714**
log2f....714
log2l....714
logb( )....**716**
logbf....716
logbl....716
logf( )....708, **718**
login shell....302
LOGIN_NAME_MAX....521, 1473
LOGIN_PROCESS....284-285

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

logl 708, 718
LOG_ 18
LOG_ constants in syslog **210**
LOG_ALERT 210
LOG_CONS 211
LOG_CRIT 210
LOG_DEBUG 210
LOG_EMERG 210
LOG_ERR 210
LOG_INFO 210
LOG_LOCAL 210
LOG_NDELAY 211
LOG_NOTICE 210
LOG_NOWAIT 211
LOG_ODELAY 211
LOG_PID 211
LOG_USER 210-211
LOG_WARNING 210
longjmp( ) **719**
LONG_MAX 1457, 1625
LONG_MIN 1457, 1625
lrand48( ) 263, **721**
lrint( ) **722**
lrintf 722
lrintl 722
lround( ) **724**
lroundf 724
lroundl 724
lsearch( ) **726**
lseek( ) **728**
lstat( ) **730**
l_ 16
L_ctermid 241
l_sysid 333
makecontext( ) **732**
malloc( ) **734**
manipulate signal sets 1354
mappings 779
MAP_ 16, 18
MAP_FAILED 779
MAP_FIXED 775, 778
MAP_PRIVATE 394, 775, 779, 783, 815
MAP_SHARED 397, 775-776
max-data-size attribute 76, 919-920
MAX_CANON 399
MAX_INPUT 399
may 2
mblen( ) **736**
mbrlen( ) **738**
mbrtowc( ) **740**
mbsinit( ) **742**
mbsrtowcs( ) **743**
mbstowcs( ) **745**
mbtowc( ) **747**
MB_CUR_MAX 736, 738, 740, 747, 1598, 1640
MC1 ..... **5**
MC2 ..... **5**
MC3 ..... **5**
MCL_ 16
MCL_CURRENT 772
MCL_FUTURE 772
memccpy( ) **749**
memchr( ) **750**
memcmp( ) **751**
memcpy( ) **752**
MEMLOCK_FUTURE 779
memmove( ) **753**
memory management 43
memory protection option 778
memset( ) **754**
message catalog descriptor 297, 302, 307
message parts 39
message priority 38
  high-priority 38
  normal 38
  priority 38
message-discard mode 1173
message-nondiscard mode 1173
MET-1MEST 1549
MF **5**
MINSIGSTKSZ 1351
mkdir( ) **755**
mkfifo( ) **758**
mknod( ) **761**
mkstemp( ) **764**
mktemp( ) **766**
mktime( ) **768**
ML **6**
mlock( ) **770**
mlockall( ) **772**
MLR **6**
mmap( ) **774**
MM_ 18
MM_APPL 385
MM_CONSOLE 385
MM_ERROR 386-387
mm_FIRM 385
MM_HALT 386
MM_HARD 385
MM_INFO 386
MM_NOCON 386
MM_NOMSG 386

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

MM_NOSEV....386
MM_NOTOK....386
MM_NRECOV....385
MM_NULLMC....385
MM_OK....386
MM_OPSYS....385
MM_PRINT....385, 387
MM_RECOVER....385
MM_SOFT....385
MM_UTIL....385
MM_WARNING....386
modf( )....**781**
modff....781
modfl....781
MON....**6**
MORECTL....525
MOREDATA....525
MPR....**6**
mprotect( )....**783**
MQ_....16
mq_....16
mq_close( )....**785**
mq_getattr( )....**786**
mq_notify( )....**788**
mq_open( )....**790**
MQ_OPEN_MAX....1473
MQ_PRIO_MAX....796-797, 1473
mq_receive( )....**793**
mq_send( )....**796**
mq_setattr( )....**798**
mq_timedreceive....793
mq_timedreceive( )....**800**
mq_timedsend....796
mq_timedsend( )....**801**
mq_unlink( )....**802**
mrand48( )....263, **804**
MSG....**6**, 18
msgctl( )....**805**
msgget( )....**807**
msgrcv( )....**809**
msgsnd( )....**812**
MSGVERB....386-387
msg_....16
MSG_....18
MSG_ANY....524
MSG_BAND....524, 1154
MSG_EOR....1393, 1395
MSG_HIPRI....524, 1154
MSG_NOERROR....809-810
msg_perm....40
msqid....40
MST7MDT....1549
msync( )....**815**
MS_....16, 18
MS_ASYNC....776, 815
MS_INVALIDATE....815-816
MS_SYNC....776, 815
multicast....67
munlock( )....770, **818**
munlockall( )....772, **819**
munmap( )....**820**
mutex attributes....1092
mutex initialization attributes....1091
mutex performance....1092
MUXID_ALL....621-622
MUXID_R....18
MX....**6**
M_....18
name information....528
name space....**14**
NAME_MAX....25, 97, 171, 185, 187, 190
....300, 319, 338, 391, 399, 430, 449
....458, 674, 683, 730, 755, 758, 762
....791, 802, 830, 840, 845, 967, 1179
....1183, 1190, 1215, 1223, 1255, 1263
....1330, 1333, 1404, 1470, 1542, 1562
....1570, 1572
NaN....123, 407, 471
nan( )....**822**
nanf....822
nanl....822
nanosleep( )....**823**
NDEBUG....21, 128
nearbyint( )....**825**
nearbyintf....825
nearbyintl....825
network interfaces....59
NEW_TIME....284-285
nextafter( )....**827**
nextafterf....827
nextafterl....827
nexttoward....827
nexttowardf....827
nexttowardl....827
nftw( )....**829**
NGROUPS_MAX....513, 1473
nice( )....**832**
NLSPATH....171
NL_....18
NL_ARGMAX....404, 435, 468, 477
NL_CAT_LOCALE....171
nl_langinfo( )....**834**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

nohup utility 303
non-local jumps 1372
non-volatile storage 452
nrand48( ) 263, **836**
ntohl( ) 587, **837**
ntohs 587, 837
NULL 215, 241, 248, 256, 261, 779, 1181
NUM_EMPL 585
NZERO 543, 832
n_ 16
OB **6**
obsolescent 180
OF **7**
OH **7**
OLD_TIME 284-285
open a file 842
open a named semaphore 1255
open a shared memory object 1330
open( ) **838**
opendir( ) **845**
openlog( ) 210, **848**
OPEN_MAX 171, 265, 280, 332, 391
....430, 463, 507, 510, 521, 551, 593
....790, 830, 840, 845, 856, 883, 886
....1473, 1531
optarg 532, 849
opterr 532, 849
optind 532, 849
option
ADV 3
AIO 3
BAR 3
BE 4
CD 4
CPT 4
CS 4
FD 4
FR 4
FSC 5
IP6 5
MC1 5
MC2 5
MC3 5
MF 5
ML 6
MLR 6
MON 6
MPR 6
MSG 6
MX 6
PIO 7
PS 7
RS 7
RTS 7
SD 7
SEM 8
SHM 8
SIO 8
SPI 8
SPN 8
SS 8
TCT 8
TEF 8
THR 9
TMO 9
TMR 9
TPI 9
TPP 9
TPS 9
TRC 9
TRI 10
TRL 10
TSA 10
TSF 10
TSH 10
TSP 10
TSS 10
TYM 10
UP 11
XSR 11
optopt 532, 535, 849
optstring 535
orphaned process group 310
O_ 18
O_ constants
used in open( ) 838
used in posix_openpt( ) 873
O_ACCMODE 329
O_APPEND 41, 116, 246, 343, 838, 1656
O_CREAT 230, 790-791, 802, 838-840
....1248, 1254, 1329-1331
O_DSYNC 108, 838-839, 1173, 1657
O_EXCL 791, 838-839, 1254, 1329-1330
O_NDELAY 1661
O_NOCTTY 839, 843, 873
O_NONBLOCK 24, 205, 327, 360, 364
....370, 416, 420, 443, 445, 525, 617
....619, 791, 793, 796, 798, 839-841
....856, 859, 1155, 1172, 1657, 1660
O_RDONLY 251, 790, 838-840, 843, 1329, 1331
O_RDWR 322, 705, 790, 838-842, 873, 1329, 1331
O_RSYNC 839, 1173

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

O_SYNC 108, 839, 1173, 1657
O_TRUNC.....230, 839, 841, 843, 1330-1331
O_WRONLY.....230, 322, 705, 790, 838-841, 843
PAGESIZE 43, 770, 816, 820, 986, 1475
PAGE_SIZE.....1475
PATH 215
PATH environment variable 305
pathconf( ).....399, **850**
PATH_MAX.....25, 97, 171, 185, 187, 190
.....251, 300, 319, 338, 391, 399, 430
.....449, 458, 494, 575, 674, 683, 730
.....755, 758, 762, 791, 802, 830, 840
.....845, 967, 1183, 1190, 1215, 1223
.....1255, 1263, 1330, 1333, 1404, 1470
.....1477, 1542, 1562, 1570, 1572
pause( ).....**851**
pclose( ).....**852**
pd_ 16
PENDIN.....19
perror( ).....**854**
persistent connection (I_PLINK).....**622**
PF_ 18
physical write 452
ph_ 16
PIO **7**
pipe 396, 843, 1660
pipe( ) **856**
PIPE_BUF 399, 1657, 1660
PIPE_MAX 1662
plain characters.....1424
POLL 18
poll( ) **858**
POLLERR 858
POLLHUP 858
POLLIN 858
POLLNVAL 859
POLLOUT 858
POLLPRI.....858
POLLRDBAND.....858
POLLRDNORM.....858
POLLWRBAND.....858
POLLWRNORM.....858
POLL_ 18
popen( ).....**862**
portability.....**3**
POSIX.....268
POSIX.1 symbols.....13
POSIX_.....15, 17
posix_.....15, 17
POSIX_ALLOC_SIZE_MIN 399
posix_fadvise( ).....**864**
POSIX_FADV_DONTNEED.....864
POSIX_FADV_NOREUSE 864
POSIX_FADV_NORMAL.....864
POSIX_FADV_RANDOM 864
POSIX_FADV_SEQUENTIAL 864
POSIX_FADV_WILLNEED.....864
posix_fallocate( ).....**866**
posix_madvise( ).....**868**
POSIX_MADV_DONTNEED 868
POSIX_MADV_NORMAL 868
POSIX_MADV_RANDOM.....868
POSIX_MADV_SEQUENTIAL.....868
POSIX_MADV_WILLNEED 868
posix_memalign( ).....**872**
posix_mem_offset( ).....**870**
posix_openpt( ) **873**
POSIX_REC_INCR_XFER_SIZE 399
POSIX_REC_MAX_XFER_SIZE 399
POSIX_REC_MIN_XFER_SIZE 399
POSIX_REC_XFER_ALIGN 399
posix_spawn( ) **875**
posix_spawnattr_destroy( ).....**890**
posix_spawnattr_getflags( ) **892**
posix_spawnattr_getpgroup( ) **894**
posix_spawnattr_getschedparam( ) **896**
posix_spawnattr_getschedpolicy( ).....**898**
posix_spawnattr_getsigdefault( ).....**900**
posix_spawnattr_getsigmask( ).....**902**
posix_spawnattr_init( ) 890, **904**
posix_spawnattr_setflags( ) 892, **905**
posix_spawnattr_setpgroup( ) 894, **906**
posix_spawnattr_setschedparam( ).....896, **907**
posix_spawnattr_setschedpolicy( ).....898, **908**
posix_spawnattr_setsigdefault( ).....900, **909**
posix_spawnattr_setsigmask( ).....902, **910**
posix_spawnp( ) 875, **911**
posix_spawn_file_actions_addclose( ).....**883**
posix_spawn_file_actions_adddup2( ) **886**
posix_spawn_file_actions_addopen( ).....883, **888**
posix_spawn_file_actions_destroy( ) **889**
posix_spawn_file_actions_init 889
POSIX_SPAWN_RESETIDS.....876, 892
POSIX_SPAWN_SETPGROUP 876, 892, 894
POSIX_SPAWN_SETSCHEDPARAM.....892, 896
POSIX_SPAWN_SETSCHEDULER 876, 892
.....896, 898
POSIX_SPAWN_SETSIGDEF.....877, 892, 900
POSIX_SPAWN_SETSIGMASK.....892, 902
POSIX_TRACE_ADD_EVENTSET.....950
POSIX_TRACE_ALL_EVENTS.....944
POSIX_TRACE_APPEND.....917, 937

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

posix_trace_attr_destroy( ).....**912**
posix_trace_attr_getclockres( ).....**914**
posix_trace_attr_getcreatetime 914
posix_trace_attr_getgenversion 914
posix_trace_attr_getinherited( ) **916**
posix_trace_attr_getlogfullpolicy 916
posix_trace_attr_getlogsize( ) **919**
posix_trace_attr_getmaxdatasize 919
posix_trace_attr_getmaxsystemeventsize 919
posix_trace_attr_getmaxusereventsize 919
posix_trace_attr_getname( ).....914, **922**
posix_trace_attr_getstreamfullpolicy( ).....916, **923**
posix_trace_attr_getstreamsize( ).....919, **924**
posix_trace_attr_init( ) 912, **925**
posix_trace_attr_setinherited( ).....916, **926**
posix_trace_attr_setlogfullpolicy 916, 926
posix_trace_attr_setlogsize( ) 919, **927**
posix_trace_attr_setmaxdatasize.....919, 927
posix_trace_attr_setname( ) 914, **928**
posix_trace_attr_setstreamfullpolicy( ).....916, **929**
posix_trace_attr_setstreamsize( ).....919, **930**
posix_trace_clear( ).....**931**
posix_trace_close( ) **933**
POSIX_TRACE_CLOSE_FOR_CHILD.....916
posix_trace_create( ) **935**
posix_trace_create_withlog.....935
POSIX_TRACE_ERROR trace event 77
posix_trace_event( ) **939**
posix_trace_eventid_equal( ) **941**
posix_trace_eventid_get_name.....941
posix_trace_eventid_open( ).....939, **943**
posix_trace_eventset_add( ).....**944**
posix_trace_eventset_del.....944
posix_trace_eventset_empty.....944
posix_trace_eventset_fill.....944
posix_trace_eventset_ismember 944
posix_trace_eventtypelist_getnext_id( ).....**946**
posix_trace_eventtypelist_rewind.....946
posix_trace_event_info structure.....74
POSIX_TRACE_FILTER trace event 77, 950
POSIX_TRACE_FLUSH 917
posix_trace_flush( ).....935, **947**
POSIX_TRACE_FLUSHING 73
POSIX_TRACE_FULL.....72-74
posix_trace_getnext_event( ) **953**
posix_trace_get_attr( ) **948**
posix_trace_get_filter( ).....**950**
posix_trace_get_status( ) 948, **952**
POSIX_TRACE_INHERITED 916
POSIX_TRACE_LOOP.....73, 916-917, 937
POSIX_TRACE_NOT_FLUSHING 73
POSIX_TRACE_NOT_FULL 72-74
POSIX_TRACE_NOT_FULL.....931
POSIX_TRACE_NOT_TRUNCATED.....75, 954
POSIX_TRACE_NO_OVERRUN.....73-74, 948
posix_trace_open( ).....933, **956**
POSIX_TRACE_OVERFLOW trace event.....77
POSIX_TRACE_OVERRUN 73-74
POSIX_TRACE_RESUME trace event 77
posix_trace_rewind 933, 956
POSIX_TRACE_RUNNING 72, 959
POSIX_TRACE_SET_EVENTSET.....950
posix_trace_set_filter( ).....950, **957**
posix_trace_shutdown( ).....935, **958**
POSIX_TRACE_START trace event 77, 959
posix_trace_start( ).....**959**
posix_trace_status_info structure.....72
posix_trace_stop.....959
POSIX_TRACE_STOP trace event.....77, 959
POSIX_TRACE_SUB_EVENTSET 950
POSIX_TRACE_SUSPENDED.....72-73, 959
POSIX_TRACE_SYSTEM_EVENTS.....944
posix_trace_timedgetnext_event( ).....953, **961**
posix_trace_trid_eventid_open( ).....941, **962**
POSIX_TRACE_TRUNCATED_READ 75, 954
POSIX_TRACE_TRUNCATED_RECORD..75, 954
posix_trace_trygetnext_event( ).....953, **963**
POSIX_TRACE_UNTIL_FULL.....73, 916-917, 937
POSIX_TRACE_USER_EVENT_MAX.....939
POSIX_TRACE_WOPID_EVENTS.....944
POSIX_TYPED_MEM_ALLOCATE.....774-775
.....870, 964, 966
POSIX_TYPED_MEM_ALLOCATE_CONTIG.....
.....774-775, 870, 964, 966
posix_typed_mem_get_info( ).....**964**
POSIX_TYPED_MEM_MAP_ALLOCATABLE.....
.....820, 966
posix_typed_mem_open( ).....**966**
pow( ) **969**
powf 969
powl.....969
pread( ).....**972**, 1172
predefined stream
standard error.....37
standard input.....37
standard output.....37
preempted thread.....1038
PRI.....18
printf( ).....404, **973**
priority 38
PRIO_ 18
PRIO_INHERIT 1088

 System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PRIO_PGRP 543
PRIO_PROCESS.....543
PRIO_USER 543
process
  concurrent execution 396
  setting real and effective user IDs.....1313
  single-threaded 396
process creation 396
process group
  orphaned 310
process group ID.....539, 1305, 1317
process ID.....310
process lifetime 670
process scheduling 44
process shared memory 1092
process synchronization 1092
process termination.....309
PROT_.....16, 18
PROT_EXEC.....774, 783
PROT_NONE.....44, 774, 783
PROT_READ.....774, 783
PROT_WRITE 774, 776, 778, 783
PS.....**7**
pselect( ).....**974**
pseudo-random sequence generation functions....
 1169
PST8PDT 1549
ps_.....16
PTHREAD_.....16
pthread_.....16
pthread_atfork( ).....**979**
pthread_attr_destroy( ).....**981**
pthread_attr_getdetachstate( ).....**984**
pthread_attr_getguardsize( ).....**986**
pthread_attr_getinheritsched( ).....**988**
pthread_attr_getschedparam( ).....**990**
pthread_attr_getschedpolicy( ).....**992**
pthread_attr_getscope( ).....**994**
pthread_attr_getstack( ).....**996**
pthread_attr_getstackaddr( ).....**998**
pthread_attr_getstacksize( ).....**1000**
pthread_attr_init( ).....981, **1002**
pthread_attr_setdetachstate( ).....984, **1003**
pthread_attr_setguardsize( ).....986, **1004**
pthread_attr_setinheritsched( ).....988, **1005**
pthread_attr_setschedparam( ).....990, **1006**
pthread_attr_setschedpolicy( ).....992, **1007**
pthread_attr_setscope( ).....994, **1008**
pthread_attr_setstack( ).....996, **1009**
pthread_attr_setstackaddr( ).....998, **1010**
pthread_attr_setstacksize( ).....1000, **1011**
pthread_barrierattr_destroy( ).....**1016**
pthread_barrierattr_getpshared( ).....**1018**
pthread_barrierattr_init( ).....1016, **1020**
pthread_barrierattr_setpshared( ).....1018, **1021**
pthread_barrier_destroy( ).....**1012**
pthread_barrier_init.....1012
PTHREAD_BARRIER_SERIAL_THREAD.....1014
pthread_barrier_wait( ).....**1014**
pthread_cancel( ).....**1022**
PTHREAD_CANCELED.....57, 1057
PTHREAD_CANCEL_ASYNCHRONOUS.....54
 1132
PTHREAD_CANCEL_DEFERRED .54, 1036, 1132
PTHREAD_CANCEL_DISABLE.....54, 1132
PTHREAD_CANCEL_ENABLE.....54, 1132
pthread_cleanup_pop( ).....**1024**
pthread_cleanup_push.....1024
pthread_condattr_destroy( ).....**1041**
pthread_condattr_getclock( ).....**1043**
pthread_condattr_getpshared( ).....**1045**
pthread_condattr_init( ).....1041, **1047**
pthread_condattr_setclock( ).....1043, **1048**
pthread_condattr_setpshared( ).....1045, **1049**
pthread_cond_broadcast( ).....**1029**
pthread_cond_destroy( ).....**1032**
pthread_cond_init 1032
PTHREAD_COND_INITIALIZER.....1032
pthread_cond_signal( ).....1029, **1035**
pthread_cond_timedwait( ).....**1036**
pthread_cond_wait 1036
pthread_create( ).....**1050**
PTHREAD_CREATE_DETACHED.....30, 984
PTHREAD_CREATE_JOINABLE.....30, 984, 1067
PTHREAD_DESTRUCTOR_ITERATIONS 1064
  .....1069, 1475
pthread_detach( ).....**1053**
pthread_equal( ).....**1055**
pthread_exit( ).....**1056**
PTHREAD_EXPLICIT_SCHED.....988
pthread_getconcurrency( ).....**1058**
pthread_getcpuclockid( ).....**1060**
pthread_getschedparam( ).....**1061**
pthread_getspecific( ).....**1064**
PTHREAD_INHERIT_SCHED.....988
pthread_join( ).....**1066**
PTHREAD_KEYS_MAX.....1069, 1475
pthread_key_create( ).....**1069**
pthread_key_delete( ).....**1073**
pthread_kill( ).....**1075**
pthread_mutexattr_destroy( ).....**1091**
pthread_mutexattr_getprioceiling( ).....**1096**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

pthread_mutexattr_getprotocol( ) **1098**
pthread_mutexattr_getpshared( ) **1100**
pthread_mutexattr_gettype( ).....**1102**
pthread_mutexattr_init( ).....1091, **1104**
pthread_mutexattr_setprioceiling( ).....1096, **1105**
pthread_mutexattr_setprotocol( ) 1098, **1106**
pthread_mutexattr_setpshared( ).....1100, **1107**
pthread_mutexattr_settype( ).....1102, **1108**
PTHREAD_MUTEX_DEFAULT 1084, 1102
pthread_mutex_destroy( ) **1076**
PTHREAD_MUTEX_ERRORCHECK...1084, 1102
pthread_mutex_getprioceiling( ).....**1081**
pthread_mutex_init( ) 1076, **1083**
PTHREAD_MUTEX_INITIALIZER.....1076, 1083
pthread_mutex_lock( ) **1084**
PTHREAD_MUTEX_NORMAL 1084, 1102
PTHREAD_MUTEX_RECURSIVE 1084
.....1102-1103
pthread_mutex_setprioceiling( ).....1081, **1087**
pthread_mutex_timedlock( ).....**1088**
pthread_mutex_trylock( ).....1084, **1090**
pthread_mutex_unlock.....1084, 1090
pthread_once( ) **1109**
PTHREAD_ONCE_INIT 1109
PTHREAD_PRIO_INHERIT.....1098
PTHREAD_PRIO_NONE.....1098
PTHREAD_PRIO_PROTECT.....1085, 1098
PTHREAD_PROCESS_PRIVATE 1018, 1045
.....1092, 1100, 1127, 1141
PTHREAD_PROCESS_SHARED 1018, 1045
.....1092, 1100, 1127, 1141
pthread_rwlockattr_destroy( ).....**1125**
pthread_rwlockattr_getpshared( ).....**1127**
pthread_rwlockattr_init( ).....1125, **1129**
pthread_rwlockattr_setpshared( ).....1127, **1130**
pthread_rwlock_destroy( ) **1111**
pthread_rwlock_rdlock( ).....1111, **1113**
pthread_rwlock_timedrdlock( ) **1115**
pthread_rwlock_timedwrlock( ) **1117**
pthread_rwlock_tryrdlock( ).....1113, **1119**
pthread_rwlock_trywrlock( ).....**1120**
pthread_rwlock_unlock( ) **1122**
pthread_rwlock_wrlock( ).....1120, **1124**
PTHREAD_SCOPE_PROCESS 52, 994
PTHREAD_SCOPE_SYSTEM.....52, 994
pthread_self( ).....**1131**
pthread_setcancelstate( ).....**1132**
pthread_setcanceltype.....1132
pthread_setconcurrency( ).....1058, **1134**
pthread_setschedparam( ).....1061, **1135**
pthread_setschedprio( ).....**1136**
pthread_setspecific( ).....1064, **1138**
pthread_sigmask( ).....**1139**
pthread_spin_destroy( ).....**1141**
pthread_spin_init 1141
pthread_spin_lock( ) **1143**
pthread_spin_trylock 1143
pthread_spin_unlock( ) **1145**
PTHREAD_STACK_MIN 996, 998, 1000, 1475
pthread_testcancel( ) 1132, **1146**
PTHREAD_THREADS_MAX 1050, 1475
ptsname( ).....**1147**
putc( ) **1148**
putchar( ) **1150**
putchar_unlocked( ).....487, **1151**
putc_unlocked( ).....487, **1149**
putenv( ) **1152**
putmsg( ) **1154**
putpmsg 1154
puts( ) **1158**
pututxline( ).....284, **1160**
putwc( ).....**1161**
putwchar( ).....**1162**
pwrite( ).....**1163**, 1656
pw_ 16
p_ 16
P_ 17
P_ALL 1595
P_PGID 1595
P_PID 1595
qsort( ) **1164**
queue a signal to a process.....1370
raise( ).....**1166**
rand( ).....**1168**
random( ).....608, **1171**
RAND_MAX 1168
rand_r.....1168
read from a file 1175
read( ) **1172**
readdir( ).....**1179**
readdir_r.....1179
readlink( ) **1183**
readv( ).....**1186**
real user ID.....98, 669
realloc( ) **1188**
realpath( ) **1190**
REALTIME.....105, 107-108, 110, 113-114
.....116, 197, 203, 337, 686, 770, 772
.....785-786, 788, 790, 793, 796, 798
.....802, 823, 1233-1237, 1240, 1248-1250
.....1252, 1254, 1257, 1261, 1263, 1265
.....1329, 1333, 1369, 1376, 1382, 1521

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

.....1524-1525
REALTIME THREADS 988, 992, 994
.....1005, 1007-1008, 1061, 1081, 1087
.....1096, 1098, 1105-1106, 1135-1136
recv( ).....**1192**
recvfrom( ).....**1194**
recvmsg( ).....**1197**
regcomp( ).....**1200**
regerror 1200
regexec 1200
regfree 1200
register fork handlers.....979
REG_.....18
REG_ constants
error return values of regcomp 1202
used in regcomp 1200
REG_BADBR 1202
REG_BADPAT.....1202
REG_BADRPT.....1202
REG_EBRACE.....1202
REG_EBRACK 1202
REG_ECOLLATE.....1202
REG_ECTYPE.....1202
REG_EESCAPE.....1202
REG_EPAREN.....1202
REG_ERANGE.....1202
REG_ESPACE.....1202
REG_ESUBREG.....1202
REG_EXTENDED.....1200
REG_ICASE 1200
REG_NEWLINE 1200
REG_NOMATCH 1202
REG_NOSUB.....1200
REG_NOTBOL.....1201
REG_NOTEOL.....1201
remainder( ) **1207**
remainderf.....1207
remainderl.....1207
remove a directory 1224
remove directory entries.....1564
remove( ).....**1209**
remque( ) 610, **1211**
remquo( ) **1212**
remquof 1212
remquol 1212
rename a file 1216
rename( ).....**1214**
rewind( ) **1218**
rewinddir( ) **1219**
re_.....16
RE_DUP_MAX.....1475
rindex( ).....**1220**
rint( ).....**1221**
RLIMIT_.....18
RLIMIT_AS 555
RLIMIT_CORE.....554
RLIMIT_CPU.....554
RLIMIT_DATA 554
RLIMIT_FSIZE 554
RLIMIT_NOFILE.....554, 556
RLIMIT_STACK.....554
rlim_ 16
RLIM_.....18
RLIM_INFINITY.....554-555
RLIM_SAVED_CUR.....555
RLIM_SAVED_MAX.....555
rmdir( ).....**1223**
RMSGD.....616
RMSGN.....616
RNORM.....616
round robin 46
round( ) **1226**
roundf 1226
roundl.....1226
routing.....59
RPROTDAT.....616
RPROTDIS 616
RPROTNORM.....616
RS **7**
RS_HIPRI.....524, 615, 1154
RTLD_ 18
RTLD_DEFAULT.....261
RTLD_GLOBAL 254, 258-259, 262
RTLD_LAZY 258, 261
RTLD_LOCAL 259
RTLD_NEXT.....261-262
RTLD_NOW 258-259
RTS.....**7**
RTSIG_MAX.....1475
RUSAGE_ 18
RUSAGE_CHILDREN.....557
RUSAGE_SELF 557
ru_ 16
s6_ 16
sa_ 16
SA_.....18
SA_NOCLDSTOP.....31, 1345, 1348
SA_NOCLDWAIT 307-308, 557, 1346, 1589
SA_NODEFER 1346
SA_ONSTACK.....297, 1345
SA_RESETHAND 148, 1345-1346
SA_RESTART 148, 977, 1345, 1359

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SA_SIGINFO.....1344-1345, 1348, 1369
scalb( ).....**1228**
scalbln( ).....**1230**
scalblnf.....1230
scalblnl.....1230
scalbn 1230
scalbnf.....1230
scalbnl.....1230
scanf( ).....435, **1232**
schedule alarm 119
scheduling documentation.....54
scheduling policy
  round robin.....46
SCHED_.....16
sched_.....16
SCHED_FIFO 42, 45, 53, 298, 394
  .....543, 832, 990, 992, 1061, 1096
  .....1113, 1238, 1257
sched_getparam( ).....**1234**
sched_getscheduler( ).....**1235**
sched_get_priority_max( ).....**1233**
sched_get_priority_min 1233
SCHED_OTHER 45, 48, 543, 992, 1061, 1238
SCHED_RR.....42, 45-46, 53, 298, 394
  .....543, 832, 990, 992, 1061, 1113, 1238, 1257
sched_rr_get_interval( ) **1236**
sched_setparam( ).....**1237**
sched_setscheduler( ).....**1240**
SCHED_SPORADIC 42, 45-46, 1113, 1238, 1257
sched_yield( ).....**1243**
SCM_.....18
SCN.....18
SD.....**7**
security considerations 191, 309, 669, 1305, 1406
seed48( ).....263, **1244**
seekdir( ).....**1245**
SEEK_CUR 330, 442, 728
SEEK_END 330, 442, 728
SEEK_GET 1218
SEEK_SET 42, 110, 116, 330, 442, 728
SEGV_.....18
select( ).....974, **1247**
SEM.....**8**
semctl( ).....**1266**
semget( ) **1269**
semid 40
semop( ).....**1272**
SEM_ 16, 18
sem_.....16
sem_close( ).....**1248**
sem_destroy( ).....**1249**
SEM_FAILED 1255
sem_getvalue( ).....**1250**
sem_init( ).....**1252**
SEM_NSEMS_MAX.....1252, 1475
sem_open( ).....**1254**
sem_perm 40
sem_post( ).....**1257**
sem_timedwait( ).....**1259**
sem_trywait( ) **1261**
SEM_UNDO 1272
sem_unlink( ).....**1263**
SEM_VALUE_MAX.....1254, 1475
sem_wait( ) 1261, **1265**
send( ).....**1277**
sendmsg( ).....**1279**
sendto( ).....**1282**
service name 426
session.....310, 669, 1305, 1317
set cancelability state 1132
set file creation mask 1555
set process group ID for job control 1305
set-group-ID 189, 304, 309, 334
set-user-ID.....304, 309, 494, 669
SETALL 1266, 1269
setbuf( ).....**1285**
setcontext( ) 491, **1286**
setegid( ) **1287**
setenv( ).....**1288**
seteuid( ).....**1290**
setgid( ) **1291**
setgrent( ) 272, **1293**
sethostent( ) 274, **1294**
setitimer( ).....519, **1295**
setjmp( ) **1296**
setkey( ).....**1298**
setlocale( ).....**1299**
setlogmask( ) 210, **1303**
setnetent( ) 276, **1304**
setpgid( ).....**1305**
setpgrp( ) **1307**
setpriority( ).....543, **1308**
setprotoent( ) 278, **1309**
setpwent( ) 280, **1310**
setregid( ).....**1311**
setreuid( ).....**1313**
setrlimit( ).....554, **1315**
setservent( ) 282, **1316**
setsid( ).....**1317**
setsockopt( ).....**1319**
setstate( ) 608, **1322**
setuid( ) **1323**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

setutxent( ).....284, **1326**
SETVAL.....1266, 1269
setvbuf( ).....**1327**
shall.....2
shell.....302, 310, 522, 539, 669, 1306, 1592
  job **669**
  login.....522
shell scripts
  exec.....303
shell, login 302
SHM.....**8**, 18
shmat( ).....**1335**
shmctl( ).....**1337**
shmdt( ).....**1339**
shmget( ).....**1340**
shmid.....40
SHMLBA 1335
shm_ 16
SHM_.....18
shm_open( ) **1329**
shm_perm 40
SHM_RDONLY.....1335
SHM_RND.....1335
shm_unlink( ).....**1333**
should.....2
shutdown( ).....**1342**
SHUT_.....18
SIGABRT 92
sigaction( ).....**1344**
sigaddset( ).....**1350**
SIGALRM 119, 519, 1388, 1551, 1568
sigaltstack( ).....**1351**
SIGBUS.....44, 776, 779, 1139
SIGCANCEL.....1022
SIGCHLD.....211, 307-308, 557, 582
  .....1345, 1348, 1357, 1481, 1589, 1595
SIGCLD.....1348
SIGCONT 34, 308, 310, 668-669
sigdelset( ) **1353**
sigemptyset( ) **1354**
SIGEV_.....16
sigev_ 16
SIGEV_NONE.....29, 42
SIGEV_SIGNAL 29, 1521
SIGEV_THREAD 29-30
sigfillset( ) **1356**
SIGFPE.....1139, 1363
sighold( ).....**1357**
SIGHUP.....205, 308, 310
sigignore.....1357
SIGILL 1139, 1363
SIGINT 396, 1481
siginterrupt( ).....**1359**
sigismember( ).....**1361**
SIGKILL.....669, 1344, 1348, 1357
siglongjmp( ).....**1362**
signal generation and delivery 28
  realtime.....29
signal handler.....1363
signal( ).....**1363**
signaling a condition 1030
signals 28
signbit( ).....**1365**
sigpause( ).....1357, **1366**
sigpending( ).....**1367**
SIGPIPE.....327, 360, 416, 420, 443, 446, 1155, 1659
SIGPOLL.....205, 614-615
sigprocmask( ).....1139, **1368**
SIGPROF 519
sigqueue( ).....**1369**
SIGQUEUE_MAX 1369, 1475
SIGQUIT.....1481
sigrelse( ) 1357, **1371**
SIGRTMAX 29-30, 1369, 1376, 1380
SIGRTMIN 29-30, 1369, 1376, 1380
SIGSEGV.....44, 555, 820, 986, 1139, 1363
sigset 1357, 1371
sigsetjmp( ).....**1372**
SIGSTKSZ 1351
SIGSTOP.....29, 1344, 1348, 1357
sigsuspend( ).....**1374**
sigtimedwait( ) **1376**
SIGTSTP.....29
SIGTTIN 29, 364, 370, 1174
SIGTTOU.....29, 327, 360, 416, 420
  .....443, 445, 1490, 1492, 1494, 1501
  .....1504, 1659
SIGURG 615
SIGVTALRM.....519
sigwait( ).....**1380**
sigwaitinfo( ) 1376, **1382**
SIGXCPU.....554
SIGXFSZ.....554, 1542
SIG_ 16, 18
SIG_BLOCK.....1139
SIG_DFL.....30, 297, 555, 1344, 1346, 1363
SIG_ERR.....148, 1363
SIG_HOLD.....1357
SIG_IGN.....30-31, 297, 303, 307-308, 557
  .....1344, 1363, 1589
SIG_SETMASK.....1139
SIG_UNBLOCK 1139

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sin( ) **1383**
sin6_ 16
sinf 1383
sinh( ) **1385**
sinhf 1385
sinhl 1385
sinl( ) 1383, **1387**
sin_ 16
SIO **8**
SIOCATMARK 1391
sival_ 16
SI_ 16, 18
si_ 16-17
SI_ASYNCIO 32
SI_MESGQ 32
SI_QUEUE 32
SI_TIMER 32
SI_USER 32
sleep( ) **1388**
sl_ 16
SND 18
SNDZERO 618
snprintf( ) 404, **1390**
SO 18
sockatmark( ) **1391**
socket I/O mode 60
socket out-of-band data 61
socket owner 60
socket queue limits 60
socket receive queue 61
socket types 59
socket( ) **1393**
socketpair( ) **1395**
sockets 58
  address families 58
  addressing 59
  asynchronous errors 62
  connection indication queue 62
  Internet Protocols 66
  IPv4 67
  IPv6 67
  local UNIX connections 66
  options 63
  pending error 60
  protocols 59
  signals 62
SOCK_ 18
SOCK_DGRAM 66, 1393, 1395
SOCK_RAW 66
SOCK_SEQPACKET 66, 1393, 1395
SOCK_STREAM 66, 1393, 1395
SPI **8**
SPN **8**
sporadic server policy
  execution capacity 46
  replenishment period 46
sprintf( ) 404, **1397**
spurious wakeup 1030
sqrt( ) **1398**
sqrtf 1398
sqrtl 1398
srand( ) 1168, **1400**
srand48( ) 263, **1401**
srandom( ) 608, **1402**
SS **8**
sscanf( ) 435, **1403**
SSIZE_MAX 793, 809, 1172, 1183, 1425, 1656
ss_ 16-17
SS_ 18
SS_DISABLE 1351-1352
SS_ONSTACK 1351
stack size 981
stat( ) **1404**
statvfs( ) 449, **1408**
stderr 1409
STDERR_FILENO 1409
stdin **1409**
STDIN_FILENO 862, 1409
stdio locking functions 375
stdio with explicit client locking 487
stdout 1409
STDOUT_FILENO 862, 1409
STR 18
strcasecmp( ) **1411**
strcat( ) **1412**
strchr( ) **1413**
strcmp( ) **1414**
strcoll( ) **1416**
strcpy( ) **1418**
strcspn( ) **1420**
strdup( ) **1421**
STREAM 617, 619, 1154, 1173, 1658
stream
  byte-oriented 36
  wide-oriented 36
STREAM head/tail **38**
stream-full-policy attribute 72-73, 76, 917
stream-min-size attribute 76, 920
STREAMS 22, 205, 319, 338, 524, 613
  ..... 628, 840-841, 858, 974
  access 39
  multiplexed 621

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

overview....38
streams....34
interaction with file descriptors....35
stream orientation....36
STREAM_MAX....342, 391, 862, 1475
strerror( )....**1422**
strerror_r....1422
strfmon( )....**1424**
strftime( )....**1428**
strlen( )....**1434**
strncasecmp( )....1411, **1436**
strncat( )....**1437**
strncmp( )....**1438**
strncpy( )....**1439**
strpbrk( )....**1440**
strptime( )....**1441**
strrchr( )....**1445**
strspn( )....**1446**
strstr( )....**1447**
strtod( )....**1448**
strtof....1448
strtoimax( )....**1452**
strtok( )....**1453**
strtok_r....1453
strtol( )....**1456**
strtold( )....1448, **1458**
strtoll( )....1456, **1459**
strtoul( )....**1460**
strtoull....1460
strtoumax( )....1452, **1463**
strxfrm( )....**1464**
str_....16
st_....16
ST_....18
ST_NOSUID....298, 449
ST_RDONLY....449
sun_....17
superuser....98, 191, 685, 1564
supplementary groups....191, 512
SVID....1372
SVR4....778, 823
sv_....16
SV_....18
swab( )....**1466**
swapcontext( )....732, **1467**
swprintf( )....468, **1468**
swscanf( )....477, **1469**
SWTCH....19
symbols
POSIX.1....13
symlink( )....**1470**
SYMLINK_MAX....399, 1470
SYMLOOP_MAX....147, 220, 320, 338, 391
....431, 450, 458, 464, 674, 762, 830
....1190, 1281, 1284, 1475, 1543, 1572
sync( )....**1472**
synchronously accept a signal....1377
sysconf( )....**1473**
syslog( )....210, **1480**
system crash....452
System III....191, 1557
system interfaces....83
system name....1557
system trace event type definitions....76
System V....119, 191, 305, 310, 332-333
....401, 539, 669, 756, 1224, 1317
....1348, 1372, 1496, 1557
system( )....**1481**
s_....16
S_....18
S_BANDURG....615
S_ERROR....614
S_HANGUP....615
S_HIPRI....614
S_IFBLK....761
S_IFCHR....761
S_IFDIR....761
S_IFIFO....761
S_IFREG....761
S_INPUT....614
S_IRGRP....323, 447, 761
S_IROTH....323, 447, 761
S_IRUSR....323, 447, 761
S_IRWXG....761
S_IRWXO....761
S_IRWXU....761
S_ISGID....187-189, 761, 1542, 1657
S_ISUID....187-188, 761, 1542, 1657
S_ISVTX....187, 761, 1215, 1224, 1562
S_IWGRP....323, 447, 761
S_IWOTH....323, 447, 761
S_IWUSR....323, 447, 761
S_IXGRP....761
S_IXOTH....761
S_IXUSR....761
S_MSG....614
S_OUTPUT....614
S_RDBAND....614-615
S_RDNORM....614
S_WRBAND....614
S_WRNORM....614
TABSIZE....150, 726

 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

tan( ).....**1485**
tanf.....1485
tanh( ).....**1487**
tanhf.....1487
tanhl.....1487
tanl( ).....1485, **1489**
tcdrain( ).....**1490**
tcflow( ).....**1492**
tcflush( ).....**1494**
tcgetattr( ).....**1496**
tcgetpgrp( ).....**1498**
tcgetsid( ).....**1500**
TCIFLUSH 1494
TCIOFF 1492
TCIOFLUSH 1494
TCION 1492
TCOFLUSH 1494
TCOOFF 1492
TCOON 1492
TCP_ 18
TCSADRAIN 1503
TCSAFLUSH 1503
TCSANOW 1503
tcsendbreak( ).....**1501**
tcsetattr( ).....**1503**
tcsetpgrp( ).....**1506**
TCT **8**
tdelete( ).....**1508**
TEF.....**8**
telldir( ).....**1512**
tempnam( ).....**1513**
terminal access control 1496, 1504
terminal device name 1546
terminate a process 309
terminology.....**1**
termios structure 1496
tfind( ).....1508, **1515**
tgamma( ).....**1516**
tgammaf 1516
tgammal.....1516
THR **9**
thread creation 1051
thread creation attributes.....981
thread ID 1055
thread IDs.....51
thread mutexes.....51
thread scheduling.....52
thread termination 1056
thread-safety.....50, 375
thread-specific data key creation 1071
thread-specific data key deletion 1073
thread-specific data management.....1065
threads 50
  regular file operations 58
time( ).....**1518**
timer ID.....1523
TIMER_ 17-18
timer_ 17
TIMER_ABSTIME.....49, 200, 1525
timer_create( ).....**1521**
timer_delete( ).....**1524**
timer_getoverrun( ).....**1525**
timer_gettime 1525
TIMER_MAX.....1475
timer_settime.....1525
times( ).....**1528**
timezone( ).....**1530**
TMO.....**9**
tmpfile( ).....**1531**
tmpnam( ).....**1533**
TMP_MAX.....1513, 1532-1533
TMR.....**9**
tms_ 16
tm_.....17
toascii( ).....**1535**
tolower( ).....**1536**
TOSTOP.....327, 360, 416, 420, 443, 445, 1658
toupper( ).....**1537**
towctrans( ).....**1538**
towlower( ).....**1539**
towupper( ).....**1540**
TPI.....**9**
TPP.....**9**
TPS **9**
trace event, POSIX_TRACE_ERROR 77
trace event, POSIX_TRACE_FILTER 77, 950
trace event, POSIX_TRACE_OVERFLOW.....77
trace event, POSIX_TRACE_RESUME 77
trace event, POSIX_TRACE_START 77, 959
trace event, POSIX_TRACE_STOP.....77, 959
trace functions 79
trace-name attribute 76, 914
TRACE_EVENT_NAME_MAX 939, 941
TRACE_SYS_MAX.....936
TRACE_USER_EVENT_MAX.....939, 941
TRACING.....912, 914, 916, 919, 922-931
  .....933, 935, 939, 941, 943-944, 946-948
  .....950, 952-953, 956-959, 961-963
TRAP_ 18
TRC **9**
TRI **10**
TRL **10**

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

trunc( ).....**1541**
truncate( ).....**1542**
truncation-status attribute.....939
truncf.....1541
truncf( ).....**1544**
truncl.....1541, 1544
TSA.....**10**
tsearch( ).....1508, **1545**
TSF.....**10**
TSH.....**10**
TSP.....**10**
TSS.....**10**
ttyname( ).....**1546**
ttyname_r.....1546
TTY_NAME_MAX.....1475, 1546
tv_.....16-17
twalk( ).....1508, **1548**
TYM.....**10**
tzname.....1549
TZNAME_MAX.....1475
tzset( ).....**1549**
t_uscalar_t.....616
ualarm( ).....**1551**
uc_.....16-17
UINT_MAX.....119, 1389
UIO_MAXIOV.....17
ulimit( ).....**1553**
ULLONG_MAX.....1461
ULONG_MAX.....1461, 1632
UL_.....17
UL_GETFSIZE.....1553
UL_SETFSIZE.....1553
umask( ).....**1555**
uname( ).....**1557**
undefined.....2
underlying function.....36
ungetc( ).....**1559**
ungetwc( ).....**1560**
unicast.....67
unlink( ).....**1562**
unlockpt( ).....**1566**
unsetenv( ).....**1567**
unspecified.....2
UP.....**11**
US-ASCII.....627
user ID
  real and effective.....1313
  setting real and effective.....1313
user trace event type definitions.....79
USER_PROCESS.....284-285
usleep( ).....**1568**
UTC.....1549
utime( ).....**1570**
utimes( ).....**1572**
utim_.....17
uts_.....17
ut_.....17
va_arg( ).....**1574**
va_copy.....1574
va_end.....1574
va_start.....1574
VDISCARD.....19
VDSUSP.....19
Version 7.....119, 191, 669, 1557
vfork( ).....**1575**
vfprintf( ).....**1577**
vfscanf( ).....**1578**
vfwprintf( ).....**1579**
vfwscanf( ).....**1580**
VISIT.....1508, 1548
VLNEXT.....19
vprintf( ).....1577, **1581**
VREPRINT.....19
vscanf( ).....1578, **1582**
vsnprintf( ).....1577, **1583**
vsprintf.....1577, 1583
vsscanf( ).....1578, **1584**
VSTATUS.....19
vswprintf( ).....1579, **1585**
vswscanf( ).....1580, **1586**
VWERASE.....19
vwprintf( ).....1579, **1587**
vwscanf( ).....1580, **1588**
wait for process termination.....1592
wait for thread termination.....1067
wait( ).....**1589**
waitid( ).....**1595**
waiting on a condition.....1037
waitpid( ).....1589, **1597**
WARNING.....386
warning
  OB.....6
  OF.....7
WCONTINUED.....1589, 1595
wcrtomb( ).....**1598**
wcscat( ).....**1600**
wcschr( ).....**1601**
wcscmp( ).....**1602**
wcscoll( ).....**1603**
wcscpy( ).....**1604**
wcscspn( ).....**1605**
wcsftime( ).....**1606**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

wcslen( ).....**1608**
wcsncat( ).....**1609**
wcsncmp( ).....**1610**
wcsncpy( ).....**1611**
wcspbrk( ).....**1612**
wcsrchr( ).....**1613**
wcsrtombs( ).....**1614**
wcsspn( ).....**1616**
wcsstr( ).....**1617**
wcstod( ).....**1618**
wcstof.....1618
wcstoimax( ).....**1621**
wcstok( ).....**1622**
wcstol( ).....**1624**
wcstold( ).....1618, **1627**
wcstoll( ).....1624, **1628**
wcstombs( ).....**1629**
wcstoul( ).....**1631**
wcstoumax( ).....1621, **1634**
wcswcs( ).....**1635**
wcswidth( ).....**1636**
wcsxfrm( ).....**1637**
wctob( ).....**1639**
wctomb( ).....**1640**
wctrans( ).....**1642**
wctype( ).....**1643**
wcwidth( ).....**1645**
WEOF.....82, 650-651, 653-654, 656-663 .....1539-1540, 1560
WEXITED.....1595
WEXITSTATUS.....1590
we_.....17
wide-oriented stream.....36
WIFCONTINUED.....1590
WIFEXITED.....1590
WIFSIGNALED.....1590
WIFSTOPPED.....1590, 1593
wmemchr( ).....**1646**
wmemcmp( ).....**1647**
wmemcpy( ).....**1648**
wmemmove( ).....**1649**
wmemset( ).....**1650**
WNOHANG.....1348, 1589, 1595
WNOWAIT.....1595
wordexp( ).....**1651**
wordfree.....1651
wprintf( ).....468, **1655**
WRDE_.....18
WRDE_APPEND.....1652
WRDE_BADCHAR.....1653
WRDE_BADVAL.....1653
WRDE_CMDSUB.....1653
WRDE_DOOFFS.....1652
WRDE_NOCMD.....1652
WRDE_NOSPACE.....1653
WRDE_REUSE.....1652
WRDE_SHOWERR.....1652
WRDE_SYNTAX.....1653
WRDE_UNDEF.....1652
write to a file.....1660
write( ).....**1656**
writev( ).....**1664**
wscanf( ).....477, **1666**
WSTOPPED.....1595
WSTOPSIG.....1590
WTERMSIG.....1590
WUNTRACED.....1589, 1592
XSI.....**11**
XSI interprocess communication.....**39**
XSR.....**11**
X_OK.....98
y0( ).....**1667**
y1.....1667
yn.....1667
zombie process.....307

System Interfaces, Issue 6 — Copyright © 2001-2003, IEEE and The Open Group. All rights reserved.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ISO/IEC 9945-2:2003(E)
System Interfaces, IEEE Std 1003.1, 2003 Edition
The Open Group Technical Standard, Issue 6

ICS 35.060

Price based on 18 pages

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale