

IEEE Std 1003.1™, 2004 Edition

The Open Group Technical Standard  
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001, IEEE Std 1003.1™-2001/Cor 1-2002  
and IEEE Std 1003.1™-2001/Cor 2-2004

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

## Base Definitions

Sponsor

**Portable Applications Standards Committee**  
of the  
**IEEE Computer Society**

and

**The Open Group**



THE *Open* GROUP

[This page intentionally left blank]

## Abstract

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

This 2004 Edition includes IEEE Std 1003.1-2001/Cor 1-2002 and IEEE Std 1003.1-2001/Cor 2-2004 incorporated into IEEE Std 1003.1-2001 (the base document). The two Corrigenda address 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<sup>™</sup>-1996, IEEE Std 1003.2<sup>™</sup>-1992, ISO/IEC 9945-1: 1996, ISO/IEC 9945-2: 1993, and the Base Specifications of The Open Group Single UNIX<sup>®</sup> 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<sup>®</sup>), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Published 30 April 2004 by the Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York, NY 10016-5997, U.S.A.  
ISBN: 0-7381-4039-2 / SH95234 PDF 0-7381-4040-6 / SS95234  
Printed in the United States of America by the IEEE.

Published 30 April 2004 by The Open Group  
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K.  
Document Number: C046  
ISBN: 1-931624-43-7  
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](mailto: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 [www.opengroup.org/austin/defectform.html](http://www.opengroup.org/austin/defectform.html).

## 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. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE. Current interpretations can be accessed at <http://standards.ieee.org/reading/ieee/interp/index.html>.

Errata, if any, for this and all other standards can be accessed at <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with the IEEE.<sup>1</sup> 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.<sup>2</sup> 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>.

The IEEE publications catalog and ordering information are available at <http://shop.ieee.org/store>.

1. For this standard, please send comments via the Austin Group as requested on page ii.

2. Please refer to the special provisions for this standard on page ii 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.

## The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX certification.

Further information on The Open Group is available at [www.opengroup.org](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 [www.opengroup.org/testing](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 [www.opengroup.org/pubs](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 [www.opengroup.org/corrigenda](http://www.opengroup.org/corrigenda).

The Open Group publications catalog and ordering information are available at [www.opengroup.org/pubs](http://www.opengroup.org/pubs).

# Contents

<b>Chapter 1</b>	<b>Introduction.....</b>	<b>1</b>
1.1	Scope.....	1
1.2	Conformance .....	4
1.3	Normative References .....	4
1.4	Terminology .....	5
1.5	Portability .....	6
1.5.1	Codes .....	6
1.5.2	Margin Code Notation.....	14
 <b>Chapter 2</b>	 <b>Conformance.....</b>	 <b>17</b>
2.1	Implementation Conformance.....	17
2.1.1	Requirements.....	17
2.1.2	Documentation.....	17
2.1.3	POSIX Conformance .....	18
2.1.3.1	POSIX System Interfaces.....	18
2.1.3.2	POSIX Shell and Utilities.....	20
2.1.4	XSI Conformance .....	21
2.1.4.1	XSI System Interfaces.....	21
2.1.4.2	XSI Shell and Utilities Conformance .....	22
2.1.5	Option Groups.....	22
2.1.5.1	Subprofiling Considerations.....	22
2.1.5.2	XSI Option Groups .....	24
2.1.6	Options.....	28
2.1.6.1	System Interfaces .....	29
2.1.6.2	Shell and Utilities.....	29
2.2	Application Conformance.....	31
2.2.1	Strictly Conforming POSIX Application.....	31
2.2.2	Conforming POSIX Application.....	32
2.2.2.1	ISO/IEC Conforming POSIX Application.....	32
2.2.2.2	<National Body> Conforming POSIX Application.....	32
2.2.3	Conforming POSIX Application Using Extensions .....	32
2.2.4	Strictly Conforming XSI Application .....	32
2.2.5	Conforming XSI Application Using Extensions.....	33
2.3	Language-Dependent Services for the C Programming Language..	33
2.4	Other Language-Related Specifications .....	33
 <b>Chapter 3</b>	 <b>Definitions.....</b>	 <b>35</b>
3.1	Abortive Release .....	35
3.2	Absolute Pathname .....	35
3.3	Access Mode .....	35
3.4	Additional File Access Control Mechanism.....	35
3.5	Address Space .....	35

3.6	Advisory Information .....	35
3.7	Affirmative Response .....	36
3.8	Alert .....	36
3.9	Alert Character (<alert>) .....	36
3.10	Alias Name .....	36
3.11	Alignment .....	36
3.12	Alternate File Access Control Mechanism .....	36
3.13	Alternate Signal Stack .....	37
3.14	Ancillary Data .....	37
3.15	Angle Brackets .....	37
3.16	Application .....	37
3.17	Application Address .....	37
3.18	Application Program Interface (API) .....	37
3.19	Appropriate Privileges .....	37
3.20	Argument .....	38
3.21	Arm (a Timer) .....	38
3.22	Asterisk .....	38
3.23	Async-Cancel-Safe Function .....	38
3.24	Asynchronous Events .....	38
3.25	Asynchronous Input and Output .....	38
3.26	Async-Signal-Safe Function .....	38
3.27	Asynchronously-Generated Signal .....	39
3.28	Asynchronous I/O Completion .....	39
3.29	Asynchronous I/O Operation .....	39
3.30	Authentication .....	39
3.31	Authorization .....	39
3.32	Background Job .....	39
3.33	Background Process .....	39
3.34	Background Process Group (or Background Job) .....	39
3.35	Backquote .....	40
3.36	Backslash .....	40
3.37	Backspace Character (<backspace>) .....	40
3.38	Barrier .....	40
3.39	Base Character .....	40
3.40	Basename .....	40
3.41	Basic Regular Expression (BRE) .....	40
3.42	Batch Access List .....	40
3.43	Batch Administrator .....	41
3.44	Batch Client .....	41
3.45	Batch Destination .....	41
3.46	Batch Destination Identifier .....	41
3.47	Batch Directive .....	41
3.48	Batch Job .....	41
3.49	Batch Job Attribute .....	42
3.50	Batch Job Identifier .....	42
3.51	Batch Job Name .....	42
3.52	Batch Job Owner .....	42
3.53	Batch Job Priority .....	42



3.54	Batch Job State .....	42
3.55	Batch Name Service.....	42
3.56	Batch Name Space .....	42
3.57	Batch Node .....	43
3.58	Batch Operator .....	43
3.59	Batch Queue.....	43
3.60	Batch Queue Attribute .....	43
3.61	Batch Queue Position .....	43
3.62	Batch Queue Priority .....	43
3.63	Batch Rerunability .....	43
3.64	Batch Restart .....	44
3.65	Batch Server .....	44
3.66	Batch Server Name .....	44
3.67	Batch Service .....	44
3.68	Batch Service Request .....	44
3.69	Batch Submission.....	44
3.70	Batch System.....	44
3.71	Batch Target User.....	45
3.72	Batch User.....	45
3.73	Bind.....	45
3.74	Blank Character (<blank>).....	45
3.75	Blank Line .....	45
3.76	Blocked Process (or Thread) .....	45
3.77	Blocking .....	45
3.78	Block-Mode Terminal.....	45
3.79	Block Special File.....	46
3.80	Braces.....	46
3.81	Brackets.....	46
3.82	Broadcast .....	46
3.83	Built-In Utility (or Built-In) .....	46
3.84	Byte .....	46
3.85	Byte Input/Output Functions .....	47
3.86	Carriage-Return Character (<carriage-return>) .....	47
3.87	Character .....	47
3.88	Character Array.....	47
3.89	Character Class.....	47
3.90	Character Set.....	47
3.91	Character Special File.....	48
3.92	Character String .....	48
3.93	Child Process .....	48
3.94	Circumflex.....	48
3.95	Clock.....	48
3.96	Clock Jump.....	48
3.97	Clock Tick.....	48
3.98	Coded Character Set.....	48
3.99	Codeset.....	49
3.100	Collating Element .....	49
3.101	Collation .....	49

3.102	Collation Sequence .....	49
3.103	Column Position .....	49
3.104	Command .....	50
3.105	Command Language Interpreter .....	50
3.106	Composite Graphic Symbol .....	50
3.107	Condition Variable .....	50
3.108	Connection .....	50
3.109	Connection Mode .....	50
3.110	Connectionless Mode .....	50
3.111	Control Character .....	51
3.112	Control Operator .....	51
3.113	Controlling Process .....	51
3.114	Controlling Terminal .....	51
3.115	Conversion Descriptor .....	51
3.116	Core File .....	51
3.117	CPU Time (Execution Time) .....	51
3.118	CPU-Time Clock .....	52
3.119	CPU-Time Timer .....	52
3.120	Current Job .....	52
3.121	Current Working Directory .....	52
3.122	Cursor Position .....	52
3.123	Datagram .....	52
3.124	Data Segment .....	52
3.125	Deferred Batch Service .....	52
3.126	Device .....	52
3.127	Device ID .....	52
3.128	Directory .....	53
3.129	Directory Entry (or Link) .....	53
3.130	Directory Stream .....	53
3.131	Disarm (a Timer) .....	53
3.132	Display .....	53
3.133	Display Line .....	53
3.134	Dollar Sign .....	53
3.135	Dot .....	53
3.136	Dot-Dot .....	54
3.137	Double-Quote .....	54
3.138	Downshifting .....	54
3.139	Driver .....	54
3.140	Effective Group ID .....	54
3.141	Effective User ID .....	54
3.142	Eight-Bit Transparency .....	54
3.143	Empty Directory .....	54
3.144	Empty Line .....	55
3.145	Empty String (or Null String) .....	55
3.146	Empty Wide-Character String .....	55
3.147	Encoding Rule .....	55
3.148	Entire Regular Expression .....	55
3.149	Epoch .....	55

3.150	Equivalence Class .....	55
3.151	Era .....	55
3.152	Event Management.....	56
3.153	Executable File.....	56
3.154	Execute .....	56
3.155	Execution Time.....	56
3.156	Execution Time Monitoring.....	56
3.157	Expand .....	56
3.158	Extended Regular Expression (ERE).....	56
3.159	Extended Security Controls .....	57
3.160	Feature Test Macro .....	57
3.161	Field .....	57
3.162	FIFO Special File (or FIFO) .....	57
3.163	File.....	57
3.164	File Description .....	58
3.165	File Descriptor .....	58
3.166	File Group Class.....	58
3.167	File Mode .....	58
3.168	File Mode Bits .....	58
3.169	Filename.....	58
3.170	Filename Portability .....	58
3.171	File Offset.....	59
3.172	File Other Class .....	59
3.173	File Owner Class .....	59
3.174	File Permission Bits.....	59
3.175	File Serial Number .....	59
3.176	File System .....	59
3.177	File Type.....	59
3.178	Filter .....	60
3.179	First Open (of a File).....	60
3.180	Flow Control .....	60
3.181	Foreground Job.....	60
3.182	Foreground Process .....	60
3.183	Foreground Process Group (or Foreground Job).....	60
3.184	Foreground Process Group ID .....	60
3.185	Form-Feed Character (<form-feed>) .....	60
3.186	Graphic Character.....	61
3.187	Group Database .....	61
3.188	Group ID.....	61
3.189	Group Name .....	61
3.190	Hard Limit.....	61
3.191	Hard Link .....	61
3.192	Home Directory.....	61
3.193	Host Byte Order .....	62
3.194	Incomplete Line.....	62
3.195	Inf .....	62
3.196	Instrumented Application.....	62
3.197	Interactive Shell.....	62

3.198	Internationalization.....	62
3.199	Interprocess Communication.....	62
3.200	Invoke.....	62
3.201	Job.....	63
3.202	Job Control .....	63
3.203	Job Control Job ID.....	63
3.204	Last Close (of a File) .....	63
3.205	Line .....	63
3.206	Linger.....	63
3.207	Link .....	63
3.208	Link Count.....	64
3.209	Local Customs .....	64
3.210	Local Interprocess Communication (Local IPC) .....	64
3.211	Locale.....	64
3.212	Localization.....	64
3.213	Login.....	64
3.214	Login Name.....	64
3.215	Map .....	64
3.216	Marked Message .....	65
3.217	Matched .....	65
3.218	Memory Mapped Files.....	65
3.219	Memory Object.....	65
3.220	Memory-Resident .....	65
3.221	Message.....	65
3.222	Message Catalog .....	66
3.223	Message Catalog Descriptor.....	66
3.224	Message Queue .....	66
3.225	Mode.....	66
3.226	Monotonic Clock.....	66
3.227	Mount Point .....	66
3.228	Multi-Character Collating Element.....	66
3.229	Mutex.....	66
3.230	Name .....	67
3.231	Named STREAM.....	67
3.232	NaN (Not a Number).....	67
3.233	Native Language.....	67
3.234	Negative Response .....	67
3.235	Network.....	67
3.236	Network Address.....	67
3.237	Network Byte Order.....	68
3.238	Newline Character (<newline>) .....	68
3.239	Nice Value .....	68
3.240	Non-Blocking.....	68
3.241	Non-Spacing Characters .....	68
3.242	NUL .....	68
3.243	Null Byte.....	69
3.244	Null Pointer.....	69
3.245	Null String.....	69

3.246	Null Wide-Character Code .....	69
3.247	Number Sign .....	69
3.248	Object File .....	69
3.249	Octet .....	69
3.250	Offset Maximum .....	69
3.251	Opaque Address .....	69
3.252	Open File .....	70
3.253	Open File Description .....	70
3.254	Operand .....	70
3.255	Operator .....	70
3.256	Option .....	70
3.257	Option-Argument .....	70
3.258	Orientation .....	70
3.259	Orphaned Process Group .....	70
3.260	Page .....	71
3.261	Page Size .....	71
3.262	Parameter .....	71
3.263	Parent Directory .....	71
3.264	Parent Process .....	71
3.265	Parent Process ID .....	71
3.266	Pathname .....	72
3.267	Pathname Component .....	72
3.268	Path Prefix .....	72
3.269	Pattern .....	72
3.270	Period .....	72
3.271	Permissions .....	72
3.272	Persistence .....	72
3.273	Pipe .....	73
3.274	Polling .....	73
3.275	Portable Character Set .....	73
3.276	Portable Filename Character Set .....	73
3.277	Positional Parameter .....	73
3.278	Preallocation .....	73
3.279	Preempted Process (or Thread) .....	74
3.280	Previous Job .....	74
3.281	Printable Character .....	74
3.282	Printable File .....	74
3.283	Priority .....	74
3.284	Priority Band .....	74
3.285	Priority Inversion .....	74
3.286	Priority Scheduling .....	74
3.287	Priority-Based Scheduling .....	75
3.288	Privilege .....	75
3.289	Process .....	75
3.290	Process Group .....	75
3.291	Process Group ID .....	75
3.292	Process Group Leader .....	75
3.293	Process Group Lifetime .....	75

3.294	Process ID .....	76
3.295	Process Lifetime .....	76
3.296	Process Memory Locking .....	76
3.297	Process Termination .....	76
3.298	Process-To-Process Communication .....	76
3.299	Process Virtual Time .....	76
3.300	Program.....	77
3.301	Protocol .....	77
3.302	Pseudo-Terminal .....	77
3.303	Radix Character.....	77
3.304	Read-Only File System .....	77
3.305	Read-Write Lock.....	77
3.306	Real Group ID.....	77
3.307	Real Time.....	78
3.308	Realtime Signal Extension.....	78
3.309	Real User ID .....	78
3.310	Record .....	78
3.311	Redirection .....	78
3.312	Redirection Operator .....	78
3.313	Reentrant Function .....	78
3.314	Referenced Shared Memory Object .....	78
3.315	Refresh.....	79
3.316	Regular Expression.....	79
3.317	Region.....	79
3.318	Regular File .....	79
3.319	Relative Pathname .....	79
3.320	Relocatable File.....	79
3.321	Relocation .....	79
3.322	Requested Batch Service.....	79
3.323	(Time) Resolution .....	79
3.324	Root Directory .....	80
3.325	Runnable Process (or Thread) .....	80
3.326	Running Process (or Thread) .....	80
3.327	Saved Resource Limits.....	80
3.328	Saved Set-Group-ID .....	80
3.329	Saved Set-User-ID.....	80
3.330	Scheduling.....	80
3.331	Scheduling Allocation Domain.....	80
3.332	Scheduling Contention Scope .....	81
3.333	Scheduling Policy.....	81
3.334	Screen .....	81
3.335	Scroll .....	81
3.336	Semaphore.....	81
3.337	Session.....	81
3.338	Session Leader .....	82
3.339	Session Lifetime .....	82
3.340	Shared Memory Object .....	82
3.341	Shell.....	82

3.342	Shell, the.....	82
3.343	Shell Script.....	82
3.344	Signal .....	82
3.345	Signal Stack .....	83
3.346	Single-Quote .....	83
3.347	Slash.....	83
3.348	Socket .....	83
3.349	Socket Address.....	83
3.350	Soft Limit .....	83
3.351	Source Code .....	83
3.352	Space Character (<space>).....	84
3.353	Spawn.....	84
3.354	Special Built-In .....	84
3.355	Special Parameter.....	84
3.356	Spin Lock .....	84
3.357	Sporadic Server .....	84
3.358	Standard Error .....	84
3.359	Standard Input.....	84
3.360	Standard Output .....	84
3.361	Standard Utilities .....	85
3.362	Stream.....	85
3.363	STREAM .....	85
3.364	STREAM End.....	85
3.365	STREAM Head .....	85
3.366	STREAMS Multiplexor .....	85
3.367	String .....	85
3.368	Subshell.....	86
3.369	Successfully Transferred.....	86
3.370	Supplementary Group ID .....	86
3.371	Suspended Job .....	86
3.372	Symbolic Link.....	86
3.373	Synchronized Input and Output .....	86
3.374	Synchronized I/O Completion.....	86
3.375	Synchronized I/O Data Integrity Completion .....	87
3.376	Synchronized I/O File Integrity Completion .....	87
3.377	Synchronized I/O Operation .....	87
3.378	Synchronous I/O Operation .....	87
3.379	Synchronously-Generated Signal.....	87
3.380	System .....	87
3.381	System Crash .....	88
3.382	System Console .....	88
3.383	System Databases .....	88
3.384	System Documentation .....	88
3.385	System Process .....	88
3.386	System Reboot .....	88
3.387	System Trace Event.....	88
3.388	System-Wide.....	89
3.389	Tab Character (<tab>) .....	89

3.390	Terminal (or Terminal Device) .....	89
3.391	Text Column.....	89
3.392	Text File .....	89
3.393	Thread .....	89
3.394	Thread ID.....	90
3.395	Thread List .....	90
3.396	Thread-Safe .....	90
3.397	Thread-Specific Data Key.....	90
3.398	Tilde .....	90
3.399	Timeouts .....	90
3.400	Timer .....	90
3.401	Timer Overrun.....	90
3.402	Token .....	91
3.403	Trace Analyzer Process .....	91
3.404	Trace Controller Process .....	91
3.405	Trace Event .....	91
3.406	Trace Event Type.....	91
3.407	Trace Event Type Mapping.....	91
3.408	Trace Filter .....	91
3.409	Trace Generation Version.....	91
3.410	Trace Log.....	91
3.411	Trace Point .....	92
3.412	Trace Stream.....	92
3.413	Trace Stream Identifier.....	92
3.414	Trace System .....	92
3.415	Traced Process .....	92
3.416	Tracing Status of a Trace Stream.....	92
3.417	Typed Memory Name Space .....	92
3.418	Typed Memory Object .....	92
3.419	Typed Memory Pool.....	92
3.420	Typed Memory Port .....	93
3.421	Unbind.....	93
3.422	Unit Data.....	93
3.423	Upshifting.....	93
3.424	User Database .....	93
3.425	User ID .....	93
3.426	User Name.....	93
3.427	User Trace Event .....	94
3.428	Utility.....	94
3.429	Variable .....	94
3.430	Vertical-Tab Character (<vertical-tab>) .....	94
3.431	White Space.....	94
3.432	Wide-Character Code (C Language) .....	94
3.433	Wide-Character Input/Output Functions.....	94
3.434	Wide-Character String .....	95
3.435	Word .....	95
3.436	Working Directory (or Current Working Directory) .....	95
3.437	Worldwide Portability Interface .....	95



	3.438	Write .....	95
	3.439	XSI .....	95
	3.440	XSI-Conformant .....	95
	3.441	Zombie Process .....	96
	3.442	$\pm 0$ .....	96
<b>Chapter</b>	<b>4</b>	<b>General Concepts .....</b>	<b>97</b>
	4.1	Concurrent Execution .....	97
	4.2	Directory Protection .....	97
	4.3	Extended Security Controls .....	97
	4.4	File Access Permissions .....	97
	4.5	File Hierarchy .....	98
	4.6	Filenames .....	98
	4.7	File Times Update .....	98
	4.8	Host and Network Byte Orders .....	99
	4.9	Measurement of Execution Time .....	99
	4.10	Memory Synchronization .....	100
	4.11	Pathname Resolution .....	100
	4.12	Process ID Reuse .....	101
	4.13	Scheduling Policy .....	101
	4.14	Seconds Since the Epoch .....	102
	4.15	Semaphore .....	102
	4.16	Thread-Safety .....	103
	4.17	Tracing .....	103
	4.18	Treatment of Error Conditions for Mathematical Functions .....	105
	4.18.1	Domain Error .....	105
	4.18.2	Pole Error .....	106
	4.18.3	Range Error .....	106
	4.18.3.1	Result Overflows .....	106
	4.18.3.2	Result Underflows .....	106
	4.19	Treatment of NaN Arguments for the Mathematical Functions .....	106
	4.20	Utility .....	107
	4.21	Variable Assignment .....	107
<b>Chapter</b>	<b>5</b>	<b>File Format Notation .....</b>	<b>109</b>
<b>Chapter</b>	<b>6</b>	<b>Character Set .....</b>	<b>113</b>
	6.1	Portable Character Set .....	113
	6.2	Character Encoding .....	116
	6.3	C Language Wide-Character Codes .....	117
	6.4	Character Set Description File .....	117
	6.4.1	State-Dependent Character Encodings .....	120
<b>Chapter</b>	<b>7</b>	<b>Locale .....</b>	<b>123</b>
	7.1	General .....	123
	7.2	POSIX Locale .....	124
	7.3	Locale Definition .....	124
	7.3.1	LC_CTYPE .....	126

7.3.1.1	LC_CTYPE Category in the POSIX Locale .....	130
7.3.2	LC_COLLATE.....	134
7.3.2.1	The collating-element Keyword .....	135
7.3.2.2	The collating-symbol Keyword .....	136
7.3.2.3	The order_start Keyword .....	136
7.3.2.4	Collation Order .....	137
7.3.2.5	The order_end Keyword .....	139
7.3.2.6	LC_COLLATE Category in the POSIX Locale .....	139
7.3.3	LC_MONETARY.....	142
7.3.3.1	LC_MONETARY Category in the POSIX Locale .....	144
7.3.4	LC_NUMERIC.....	145
7.3.4.1	LC_NUMERIC Category in the POSIX Locale .....	146
7.3.5	LC_TIME.....	147
7.3.5.1	LC_TIME Locale Definition.....	147
7.3.5.2	LC_TIME C-Language Access .....	149
7.3.5.3	LC_TIME Category in the POSIX Locale .....	150
7.3.6	LC_MESSAGES .....	152
7.3.6.1	LC_MESSAGES Category in the POSIX Locale .....	153
7.4	Locale Definition Grammar .....	153
7.4.1	Locale Lexical Conventions.....	153
7.4.2	Locale Grammar.....	154
<b>Chapter 8</b>	<b>Environment Variables .....</b>	<b>161</b>
8.1	Environment Variable Definition .....	161
8.2	Internationalization Variables .....	162
8.3	Other Environment Variables.....	165
<b>Chapter 9</b>	<b>Regular Expressions .....</b>	<b>169</b>
9.1	Regular Expression Definitions .....	169
9.2	Regular Expression General Requirements.....	170
9.3	Basic Regular Expressions .....	171
9.3.1	BREs Matching a Single Character or Collating Element.....	171
9.3.2	BRE Ordinary Characters.....	171
9.3.3	BRE Special Characters.....	171
9.3.4	Periods in BREs.....	172
9.3.5	RE Bracket Expression .....	172
9.3.6	BREs Matching Multiple Characters.....	174
9.3.7	BRE Precedence .....	175
9.3.8	BRE Expression Anchoring.....	175
9.4	Extended Regular Expressions .....	175
9.4.1	EREs Matching a Single Character or Collating Element.....	176
9.4.2	ERE Ordinary Characters.....	176
9.4.3	ERE Special Characters.....	176
9.4.4	Periods in EREs.....	177
9.4.5	ERE Bracket Expression.....	177
9.4.6	EREs Matching Multiple Characters.....	177
9.4.7	ERE Alternation.....	178
9.4.8	ERE Precedence .....	178

	9.4.9	ERE Expression Anchoring.....	178
	9.5	Regular Expression Grammar.....	179
	9.5.1	BRE/ERE Grammar Lexical Conventions.....	179
	9.5.2	RE and Bracket Expression Grammar.....	180
	9.5.3	ERE Grammar.....	182
<b>Chapter</b>	<b>10</b>	<b>Directory Structure and Devices .....</b>	<b>185</b>
	10.1	Directory Structure and Files .....	185
	10.2	Output Devices and Terminal Types.....	185
<b>Chapter</b>	<b>11</b>	<b>General Terminal Interface.....</b>	<b>187</b>
	11.1	Interface Characteristics.....	187
	11.1.1	Opening a Terminal Device File .....	187
	11.1.2	Process Groups.....	187
	11.1.3	The Controlling Terminal.....	188
	11.1.4	Terminal Access Control.....	188
	11.1.5	Input Processing and Reading Data.....	189
	11.1.6	Canonical Mode Input Processing .....	189
	11.1.7	Non-Canonical Mode Input Processing.....	190
	11.1.8	Writing Data and Output Processing .....	191
	11.1.9	Special Characters.....	191
	11.1.10	Modem Disconnect.....	192
	11.1.11	Closing a Terminal Device File .....	192
	11.2	Parameters that Can be Set .....	193
	11.2.1	The termios Structure .....	193
	11.2.2	Input Modes.....	193
	11.2.3	Output Modes .....	194
	11.2.4	Control Modes.....	196
	11.2.5	Local Modes .....	197
	11.2.6	Special Control Characters .....	198
<b>Chapter</b>	<b>12</b>	<b>Utility Conventions.....</b>	<b>201</b>
	12.1	Utility Argument Syntax.....	201
	12.2	Utility Syntax Guidelines.....	203
<b>Chapter</b>	<b>13</b>	<b>Headers.....</b>	<b>205</b>
	13.1	Format of Entries.....	205
		< aio.h> .....	206
		<arpa/inet.h> .....	208
		<assert.h> .....	209
		<complex.h> .....	210
		<cpio.h>.....	213
		<ctype.h>.....	214
		<dirent.h>.....	216
		<dlfcn.h> .....	218
		<errno.h>.....	219
		<fcntl.h> .....	223
		<fenv.h>.....	226

<code>&lt;float.h&gt;</code> .....	229
<code>&lt;fmtmsg.h&gt;</code> .....	233
<code>&lt;fnmatch.h&gt;</code> .....	235
<code>&lt;ftw.h&gt;</code> .....	236
<code>&lt;glob.h&gt;</code> .....	238
<code>&lt;grp.h&gt;</code> .....	240
<code>&lt;iconv.h&gt;</code> .....	241
<code>&lt;inttypes.h&gt;</code> .....	242
<code>&lt;iso646.h&gt;</code> .....	244
<code>&lt;langinfo.h&gt;</code> .....	245
<code>&lt;libgen.h&gt;</code> .....	248
<code>&lt;limits.h&gt;</code> .....	249
<code>&lt;locale.h&gt;</code> .....	264
<code>&lt;math.h&gt;</code> .....	266
<code>&lt;monetary.h&gt;</code> .....	273
<code>&lt;mqueue.h&gt;</code> .....	274
<code>&lt;ndbm.h&gt;</code> .....	276
<code>&lt;net/if.h&gt;</code> .....	277
<code>&lt;netdb.h&gt;</code> .....	278
<code>&lt;netinet/in.h&gt;</code> .....	282
<code>&lt;netinet/tcp.h&gt;</code> .....	286
<code>&lt;nl_types.h&gt;</code> .....	287
<code>&lt;poll.h&gt;</code> .....	288
<code>&lt;pthread.h&gt;</code> .....	290
<code>&lt;pwd.h&gt;</code> .....	295
<code>&lt;regex.h&gt;</code> .....	296
<code>&lt;sched.h&gt;</code> .....	298
<code>&lt;search.h&gt;</code> .....	300
<code>&lt;semaphore.h&gt;</code> .....	302
<code>&lt;setjmp.h&gt;</code> .....	303
<code>&lt;signal.h&gt;</code> .....	304
<code>&lt;spawn.h&gt;</code> .....	312
<code>&lt;stdarg.h&gt;</code> .....	314
<code>&lt;stdbool.h&gt;</code> .....	316
<code>&lt;stddef.h&gt;</code> .....	317
<code>&lt;stdint.h&gt;</code> .....	318
<code>&lt;stdio.h&gt;</code> .....	325
<code>&lt;stdlib.h&gt;</code> .....	329
<code>&lt;string.h&gt;</code> .....	333
<code>&lt;strings.h&gt;</code> .....	335
<code>&lt;stropts.h&gt;</code> .....	336
<code>&lt;sys/ipc.h&gt;</code> .....	341
<code>&lt;sys/mman.h&gt;</code> .....	343
<code>&lt;sys/msg.h&gt;</code> .....	346
<code>&lt;sys/resource.h&gt;</code> .....	347
<code>&lt;sys/select.h&gt;</code> .....	349
<code>&lt;sys/sem.h&gt;</code> .....	351
<code>&lt;sys/shm.h&gt;</code> .....	353

<sys/socket.h>.....	355
<sys/stat.h> .....	360
<sys/statvfs.h> .....	365
<sys/time.h> .....	367
<sys/timeb.h>.....	369
<sys/times.h> .....	370
<sys/types.h>.....	371
<sys/uio.h>.....	374
<sys/un.h>.....	375
<sys/utsname.h>.....	376
<sys/wait.h> .....	377
<syslog.h>.....	379
<tar.h>.....	381
<termios.h>.....	383
<tgmath.h>.....	389
<time.h> .....	393
<trace.h>.....	397
<ucontext.h>.....	401
<ulimit.h> .....	402
<unistd.h>.....	403
<utime.h>.....	423
<utmpx.h>.....	424
<wchar.h> .....	426
<wctype.h> .....	430
<wordexp.h> .....	432
<b>Index</b> .....	<b>435</b>

## List of Tables

3-1	Job Control Job ID Formats.....	63
5-1	Escape Sequences and Associated Actions .....	110
6-1	Portable Character Set.....	113
6-2	Control Character Set.....	118
7-1	Valid Character Class Combinations.....	130
10-1	Control Character Names .....	186

# *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.

# *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.<sup>3</sup> 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.

## 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,<sup>4</sup> 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.



- 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 1003.1-2001/Cor 2-2004 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

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

### This Volume

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

This volume is structured as follows:

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

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	<b>aiocb</b>	
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	<b>long</b>	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	

Reference	Example	Notes
C-Language Function Argument	<i>arg1</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<b>&lt;sys/stat.h&gt;</b>	
C-Language Keyword	<b>return</b>	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	INET_ADDRSTRLEN	
C-Language Preprocessing Directive	<b>#define</b>	
Commands within a Utility	<b>a, c</b>	
Conversion Specification, Specifier/Modifier Character	%A, g, E	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	<b>Hello, World</b>	
Filename	<i>/tmp</i>	
Literal Character	'c', '\r', '\\'	2
Literal String	"abcde"	2
Optional Items in Utility Syntax	[ ]	
Parameter	<i>&lt;directory pathname&gt;</i>	
Special Character	<i>&lt;newline&gt;</i>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	
User Input and Example Code	echo Hello, World	5
Utility Name	<i>awk</i>	
Utility Operand	<i>file_name</i>	
Utility Option	<b>-c</b>	
Utility Option with Option-Argument	<b>-w width</b>	

**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 1.5.1 (on page 6).

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  $a$ .

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

# *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

### 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

### **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



### 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.

### 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

Mark D. Bowman

Gary R. Engmann

Harold E. Epstein

H. Landis Floyd

Jay Forster\*

Howard M. Frazier

Ruben D. Garzon

James H. Gurney

Richard J. Holleman

Lowell G. Johnson

Robert J. Kennelly

Joseph L. Koepfinger\*

Peter H. Lips

L. Bruce McClung

Daleep C. Mohla

James W. Moore

Robert F. Munzner

Ronald C. Petersen

Gerald H. Peterson

John B. Posey

Gary S. Robinson

Akio Tojo

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

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

**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

### **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

**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

### 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

H. Stephen Berger

Clyde R. Camp

Richard DeBlasio

Harold E. Epstein

Julian Forster\*

Howard M. Frazier

Toshio Fukuda

Arnold M. Greenspan

Raymond Hapeman

Donald M. Heirman

Richard H. Hulett

Lowell G. Johnson

Joseph L. Koepfinger\*

Peter H. Lips

Nader Mehravari

Daleep C. Mohla

William J. Moylan

Malcolm V. Thaden

Geoffrey O. Thompson

Howard L. Wolfman

Don Wright

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

IEEE Std 1003.1-2001/Cor 2-2004 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**

Jay Ashford	Clive D.W. Feather	Rajesh Moorkath
Julian Blake	Yaacov Fenster	Peter Petrov
Mitchell Bonnett	Mark Funkenhauser	Franklin Prindle
Andries Brouwer	Ernesto Garcia	Vikram Punj
Mark Brown	Andrew Gollan	Eusebio Rufian-Zilbermann
Paul Buerger	Michael Gonzalez	Joerg Schilling
Dave Butenhof	Jean-Denis Gorin	Stephen Schwarm
Keith Chow	Matthew Gream	Gil Shultz
Geoff Clare	Scott Gudgel	Keld Simonsen
Donald W. Cragun	Bruno Haible	Nicholas Stoughton
Lee Damico	Charles Hammons	Alexander Terekhov
Maulik Dave	Barry Hedquist	Donn Terry
Juan A. de la Puente	Jon Hitchcock	Mark-Rene Uchida
Guru Dutt Dhingra	Lowell G. Johnson	Thomas Unsicker
Loic Domaine	Andrew Josey	Scott Valcourt
Ulrich Drepper	Piotr Karocki	Mats Wichmann
Sourav Dutta	David Leciston	Garrett A. Wollman
Larry Dwyer	Ryan Madron	Oren Yuen
Paul Eggert	Roger J. Martin	Mark Ziegast
Joanna Farley	George Miao	



### Austin Group Working Group Members

Harold C. Adams  
Jay Ashford  
Theodore P. Baker  
David J. Blackwood  
Julian Blake  
Mitchell Bonnett  
Andries Brouwer  
Mark Brown  
Paul Buerger  
Alan Burns  
Dave Butenhof  
Keith Chow  
Geoff Clare  
Donald W. Cragun  
Dragan Cvetkovic  
Lee Damico  
Maulik Dave  
Juan A. de la Puente  
Loic Demaille  
Steven J. Dovich  
Ulrich Drepper  
Guru Dutt Dhingra  
Sourav Dutta  
Larry Dwyer  
Paul Eggert  
Joanna Farley  
Clive D.W. Feather  
Yaacov Fenster

Mark Funkenhauser  
Ernesto Garcia  
Lois Goldthwaite  
Andrew Gollan  
Michael Gonzalez  
Karen D. Gordon  
Jean-Denis Gorin  
Matthew Gream  
Scott Gudgel  
Joseph M. Gwinn  
Steven A. Haaser  
Charles Hammons  
Ben Harris  
Barry Hedquist  
Karl Heubaum  
Jon Hitchcock  
Andreas Jaeger  
Lowell G. Johnson  
Andrew Josey  
Piotr Karocki  
Kenneth Lang  
Pi-Cheng Law  
David Leciston  
Wojtek Lerch  
Jonathan Lennox  
Nick Maclaren  
Ryan Madron  
Roger J. Martin

George Miao  
Rajesh Moorkath  
Vilhelm Mueller  
Joseph S. Myers  
Peter Petrov  
Franklin Prindle  
Vikram Punj  
Kenneth Raeburn  
Tim Robbins  
Curtis Royster Jr.  
Eusebio Rufian-Zilbermann  
Joerg Schilling  
Stephen Schwarm  
Glen Seeds  
Gil Shultz  
Keld Simonsen  
Nicholas Stoughton  
Alexander Terekhov  
Donn Terry  
Mark-Rene Uchida  
Thomas Unsicker  
Scott Valcourt  
Mats Wichmann  
Mike Wilson  
Garrett A. Wollman  
Oren Yuen  
Mark Ziegast  
Jason Zions

### **The Open Group**

When The Open Group approved the Base Specifications, Issue 6, Technical Corrigendum 2 on 18 December 2003, the membership of The Open Group Base Working Group was as follows:

**Andrew Josey**, Chair

**Mark Brown**, Austin Group Liaison

**Cathy Fox**, Technical Editor

### **Base Working Group Members**

Mark Brown	IBM Corporation
Dave Butenhof	Hewlett-Packard Company
Donald W. Cragun	Sun Microsystems, Inc.
Larry Dwyer	Hewlett-Packard Company
Ulrich Drepper	Red Hat, Inc.
Joanna Farley	Sun Microsystems, Inc.
Andrew Gollan	Sun Microsystems, Inc.
Andrew K. Roach	Sun Microsystems, Inc.
Curtis Royster Jr.	US DoD DISA
Nicholas Stoughton	USENIX Association
Kenjiro Tsuji	Sun Microsystems, Inc.

### IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001/Cor 2-2004 on 9 February 2004, 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 2-2004. Balloters may have voted for approval, disapproval, or abstention:

Jay Ashford

Julian Blake

Mark Brown

Keith Chow

Donald W. Cragun

Juan A. de la Puente

Guru Dutt Dhingra

Ernesto Garcia

Michael Gonzalez

Jean-Denis Gorin

Matthew Gream

Scott Gudgel

Charles Hammons

Barry Hedquist

Andrew Josey

Piotr Karocki

David Leciston

Ryan Madron

Roger J. Martin

George Miao

Rajesh Moorkath

Peter Petrov

Vikram Punj

Eusebio Rufian-Zilbermann

Stephen Schwarm

Gil Shultz

Mark-Rene Uchida

Thomas Unsicker

Scott Valcourt

Oren Yuen

## **IEEE-SA Standards Board**

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001/Cor 2-2004 on 9 February 2004, the membership was as follows:

**Don Wright**, Chair

**Judith Gorman**, Secretary

Chuck Adams

H. Stephen Berger

Mark D. Bowman

Joseph A. Bruder

Bob Davis

Roberto de Boisson

Julian Forster\*

Arnold M. Greenspan

Mark S. Halpin

Raymond Hapeman

Richard J. Holleman

Richard H. Hulett

Lowell G. Johnson

Hermann Koch

Joseph L. Koepfinger\*

Thomas J. McGean

Steve M. Mills

Daleep C. Mohla

Paul Nikolich

T. W. Olsen

Ronald C. Petersen

Gary S. Robinson

Frank Stone

Malcolm V. Thaden

Doug Topping

Joe D. Watson

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

**Satish K. Aggarwal**, NRC Representative

**Richard DeBlasio**, DOE Representative

**Alan Cookson**, NIST Representative

**Savoula Amanatidis**, IEEE Standards Managing Editor

---

\* Member Emeritus

# *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<sup>™</sup> is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

AIX<sup>®</sup> is a registered trademark of IBM Corporation.

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

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

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

IBM<sup>®</sup> is a registered trademark of International Business Machines Corporation.

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

All other trademarks are the property of their respective owners.

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

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

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

# *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.

# Referenced Documents

## Normative References

Normative references for this standard are defined in Section 1.3 (on page 4).

## 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.

- 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).



## Referenced Documents

### 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 11: Latin/Thai Alphabet

Part 13: Latin Alphabet No. 7

Part 14: Latin Alphabet No. 8

Part 15: Latin Alphabet No. 9

Part 16: Latin Alphabet No. 10

### 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).

#### 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.

## Referenced Documents

### 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.

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.

### **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).



# Introduction

## 1.1 Scope

IEEE Std 1003.1-2001 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. It is intended to be used by both applications developers and system implementors.

IEEE Std 1003.1-2001 comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of IEEE Std 1003.1-2001, including utility conventions and C-language header definitions, are included in the Base Definitions volume of IEEE Std 1003.1-2001.
2. 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 of IEEE Std 1003.1-2001.
3. 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 of IEEE Std 1003.1-2001.
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of IEEE Std 1003.1-2001 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of IEEE Std 1003.1-2001.

The following areas are outside of the scope of IEEE Std 1003.1-2001:

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

IEEE Std 1003.1-2001 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.

The facilities provided in IEEE Std 1003.1-2001 are drawn from the following base documents:

- IEEE Std 1003.1-1996 (POSIX-1) (incorporating IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995)
- The following amendments to the POSIX.1-1990 standard:
  - IEEE P1003.1a draft standard (Additional System Services)
  - IEEE Std 1003.1d-1999 (Additional Realtime Extensions)

- IEEE Std 1003.1g-2000 (Protocol-Independent Interfaces (PII))
- IEEE Std 1003.1j-2000 (Advanced Realtime Extensions)
- IEEE Std 1003.1q-2000 (Tracing)
- IEEE Std 1003.2-1992 (POSIX-2) (includes IEEE Std 1003.2a-1992)
- The following amendments to the ISO POSIX-2: 1993 standard:
  - IEEE P1003.2b draft standard (Additional Utilities)
  - IEEE Std 1003.2d-1994 (Batch Environment)
- Open Group Technical Standard, February 1997, System Interface Definitions, Issue 5 (XBD5) (ISBN: 1-85912-186-1, C605)
- Open Group Technical Standard, February 1997, Commands and Utilities, Issue 5 (XCU5) (ISBN: 1-85912-191-8, C604)
- Open Group Technical Standard, February 1997, System Interfaces and Headers, Issue 5 (XSH5) (in 2 Volumes) (ISBN: 1-85912-181-0, C606)
- Note:** XBD5, XCU5, and XSH5 are collectively referred to as the *Base Specifications*.
- Open Group Technical Standard, January 2000, Networking Services, Issue 5.2 (XNS5.2) (ISBN: 1-85912-241-8, C808)
- ISO/IEC 9899: 1999, Programming Languages — C.

IEEE Std 1003.1-2001 uses the Base Specifications as its organizational basis and adds the following additional functionality to them, drawn from the base documents above:

- Normative text from the ISO POSIX-1: 1996 standard and the ISO POSIX-2: 1993 standard not included in the Base Specifications
- The amendments to the POSIX.1-1990 standard and the ISO POSIX-2: 1993 standard listed above, except for parts of IEEE Std 1003.1g-2000
- Portability Considerations
- Additional rationale and notes

The following features, marked legacy or obsolescent in the base documents, are not carried forward into IEEE Std 1003.1-2001. Other features from the base documents marked legacy or obsolescent are carried forward unless otherwise noted.

From XSH5, the following legacy interfaces, headers, and external variables are not carried forward:

<code>advance()</code> , <code>brk()</code> , <code>chroot()</code> , <code>compile()</code> , <code>cuserid()</code> , <code>gamma()</code> , <code>getdtablesize()</code> , <code>getpagesize()</code> , <code>getpass()</code> ,	1
<code>getw()</code> , <code>putw()</code> , <code>re_comp()</code> , <code>re_exec()</code> , <code>regcmp()</code> , <code>regex()</code> , <code>sbrk()</code> , <code>sigstack()</code> , <code>step()</code> , <code>ttyslot()</code> ,	
<code>valloc()</code> , <code>wait3()</code> , <code>&lt;re_comp.h&gt;</code> , <code>&lt;regexp.h&gt;</code> , <code>&lt;varargs.h&gt;</code> , <code>loc1</code> , <code>__loc1</code> , <code>loc2</code> , <code>locs</code>	1

From XCU5, the following legacy utilities are not carried forward:

`calendar`, `cancel`, `cc`, `col`, `cpio`, `cu`, `dircmp`, `dis`, `egrep`, `fgrep`, `line`, `lint`, `lpstat`, `mail`, `pack`, `pcat`, `pg`, `spell`, `sum`, `tar`, `unpack`, `uulog`, `uname`, `uupick`, `uuto`

In addition, legacy features within non-legacy reference pages (for example, headers) are not carried forward.

From the ISO POSIX-1: 1996 standard, the following obsolescent features are not carried forward:



78	Page 112, CLK_TCK
79	Page 197 <i>tcgetattr()</i> rate returned option
80	From the ISO POSIX-2: 1993 standard, obsolescent features within the following pages are not
81	carried forward:
82	Page 75, zero-length prefix within <i>PATH</i>
83	Page 156, 159 <i>set</i>
84	Page 178, <i>awk</i> , use of no argument and no parentheses with length
85	Page 259, <i>ed</i>
86	Page 272, <i>env</i>
87	Page 282, <i>find</i> <b>perm</b> [-]onum
88	Page 295-296, <i>egrep</i>
89	Page 299-300, <i>head</i>
90	Page 305-306, <i>join</i>
91	Page 309-310, <i>kill</i>
92	Page 431-433, 435-436, <i>sort</i>
93	Page 444-445, <i>tail</i>
94	Page 453, 455-456, <i>touch</i>
95	Page 464-465, <i>tty</i>
96	Page 472, <i>uniq</i>
97	Page 515-516, <i>ex</i>
98	Page 542-543, <i>expand</i>
99	Page 563-565, <i>more</i>
100	Page 574-576, <i>newgrp</i>
101	Page 578, <i>nice</i>
102	Page 594-596, <i>renice</i>
103	Page 597-598, <i>split</i>
104	Page 600-601, <i>strings</i>
105	Page 624-625, <i>vi</i>
106	Page 693, <i>lex</i>
107	The <i>c89</i> utility (which specified a compiler for the C Language specified by the
108	ISO/IEC 9899: 1990 standard) has been replaced by a <i>c99</i> utility (which specifies a compiler for
109	the C Language specified by the ISO/IEC 9899: 1999 standard).
110	From XSH5, text marked OH (Optional Header) has been reviewed on a case-by-case basis and
111	removed where appropriate. The XCU5 text marked OF (Output Format Incompletely Specified)
112	and UN (Possibly Unsupportable Feature) has been reviewed on a case-by-case basis and
113	removed where appropriate.
114	For the networking interfaces, the base document is the XNS, Issue 5.2 specification. The
115	following parts of the XNS, Issue 5.2 specification are out of scope and not included in
116	IEEE Std 1003.1-2001:
117	• Part 3 (XTI)
118	• Part 4 (Appendixes)
119	Since there is much duplication between the XNS, Issue 5.2 specification and
120	IEEE Std 1003.1g-2000, material only from the following sections of IEEE Std 1003.1g-2000 has
121	been included:
122	• General terms related to sockets (Section 2.2.2)
123	• Socket concepts (Sections 5.1 through 5.3, inclusive)

- The *pselect()* function (Sections 6.2.2.1 and 6.2.3)
- The *sockatmark()* function (Section 5.4.13)
- The `<sys/select.h>` header (Section 6.2)

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

- Issues raised by IEEE or ISO/IEC Interpretations against IEEE Std 1003.1 and IEEE Std 1003.2
- Issues raised in corrigenda for the Base Specifications and working group resolutions from The Open Group
- Corrigenda and resolutions passed by The Open Group for the XNS, Issue 5.2 specification
- Changes to make the text self-consistent with the additional material merged
- A reorganization of the options in order to facilitate profiling, both for smaller profiles such as IEEE Std 1003.13, and larger profiles such as the Single UNIX Specification
- Alignment with the ISO/IEC 9899:1999 standard

## 1.2 Conformance

Conformance requirements for IEEE Std 1003.1-2001 are defined in Chapter 2 (on page 17).

## 1.3 Normative References

The following standards contain provisions which, through references in IEEE Std 1003.1-2001, constitute provisions of IEEE Std 1003.1-2001. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on IEEE Std 1003.1-2001 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ANS X3.9-1978

(Reaffirmed 1989) American National Standard for Information Systems: Standard X3.9-1978, Programming Language FORTRAN.<sup>1</sup>

ISO/IEC 646:1991

ISO/IEC 646:1991, Information Processing — ISO 7-Bit Coded Character Set for Information Interchange.<sup>2</sup>

ISO 4217:2001

ISO 4217:2001, Codes for the Representation of Currencies and Funds.

ISO 8601:2000

ISO 8601:2000, Data Elements and Interchange Formats — Information Interchange —

- 
1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018, U.S.A.
  2. ISO/IEC documents can be obtained from the ISO office: 1 Rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse

- 161 Representation of Dates and Times.
- 162 ISO C (1999)
- 163 ISO/IEC 9899: 1999, Programming Languages — C, including Technical Corrigendum 1. 1
- 164 ISO/IEC 10646-1: 2000
- 165 ISO/IEC 10646-1: 2000, Information Technology — Universal Multiple-Octet Coded
- 166 Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

## 167 1.4 Terminology

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

### 169 **can**

170 Describes a permissible optional feature or behavior available to the user or application. The

171 feature or behavior is mandatory for an implementation that conforms to

172 IEEE Std 1003.1-2001. An application can rely on the existence of the feature or behavior.

### 173 **implementation-defined**

174 Describes a value or behavior that is not defined by IEEE Std 1003.1-2001 but is selected by

175 an implementor. The value or behavior may vary among implementations that conform to

176 IEEE Std 1003.1-2001. An application should not rely on the existence of the value or

177 behavior. An application that relies on such a value or behavior cannot be assured to be

178 portable across conforming implementations.

179 The implementor shall document such a value or behavior so that it can be used correctly

180 by an application.

### 181 **legacy**

182 Describes a feature or behavior that is being retained for compatibility with older

183 applications, but which has limitations which make it inappropriate for developing portable

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

185 functionality.

### 186 **may**

187 Describes a feature or behavior that is optional for an implementation that conforms to

188 IEEE Std 1003.1-2001. An application should not rely on the existence of the feature or

189 behavior. An application that relies on such a feature or behavior cannot be assured to be

190 portable across conforming implementations.

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

### 192 **shall**

193 For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or

194 behavior that is mandatory. An application can rely on the existence of the feature or

195 behavior.

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

### 197 **should**

198 For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or

199 behavior that is recommended but not mandatory. An application should not rely on the

200 existence of the feature or behavior. An application that relies on such a feature or behavior

201 cannot be assured to be portable across conforming implementations.

202 For an application, describes a feature or behavior that is recommended programming

203 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.

**1.5 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 1.5.1). For maximum portability, an application should avoid such functionality.

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

**1.5.1 Codes**

The codes and their meanings are as follows. See also Section 1.5.2 (on page 14).

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.

246	BAR	<b>Barriers</b>
247		The functionality described is optional. The functionality described is also an extension to the
248		ISO C standard.
249		Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
250		Where additional semantics apply to a function, the material is identified by use of the BAR
251		margin legend.
252	BE	<b>Batch Environment Services and Utilities</b>
253		The functionality described is optional.
254		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
255		Where additional semantics apply to a utility, the material is identified by use of the BE margin
256		legend.
257	CD	<b>C-Language Development Utilities</b>
258		The functionality described is optional.
259		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
260		Where additional semantics apply to a utility, the material is identified by use of the CD margin
261		legend.
262	CPT	<b>Process CPU-Time Clocks</b>
263		The functionality described is optional. The functionality described is also an extension to the
264		ISO C standard.
265		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
266		Where additional semantics apply to a function, the material is identified by use of the CPT
267		margin legend.
268	CS	<b>Clock Selection</b>
269		The functionality described is optional. The functionality described is also an extension to the
270		ISO C standard.
271		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
272		Where additional semantics apply to a function, the material is identified by use of the CS
273		margin legend.
274	CX	<b>Extension to the ISO C standard</b>
275		The functionality described is an extension to the ISO C standard. Application writers may make
276		use of an extension as it is supported on all IEEE Std 1003.1-2001-conforming systems.
277		With each function or header from the ISO C standard, a statement to the effect that “any
278		conflict is unintentional” is included. That is intended to refer to a direct conflict.
279		IEEE Std 1003.1-2001 acts in part as a profile of the ISO C standard, and it may choose to further
280		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
281		conflicts.
282		Where additional semantics apply to a function or header, the material is identified by use of the
283		CX margin legend.
284	FD	<b>FORTTRAN Development Utilities</b>
285		The functionality described is optional.
286		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
287		Where additional semantics apply to a utility, the material is identified by use of the FD margin
288		legend.
289	FR	<b>FORTTRAN Runtime Utilities</b>
290		The functionality described is optional.

291		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.	
292		Where additional semantics apply to a utility, the material is identified by use of the FR margin	
293		legend.	
294	FSC	<b>File Synchronization</b>	
295		The functionality described is optional. The functionality described is also an extension to the	
296		ISO C standard.	
297		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.	
298		Where additional semantics apply to a function, the material is identified by use of the FSC	
299		margin legend.	
300	IP6	<b>IPV6</b>	
301		The functionality described is optional. The functionality described is also an extension to the	
302		ISO C standard.	
303		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.	
304		Where additional semantics apply to a function, the material is identified by use of the IP6	
305		margin legend.	
306	MC1	<b>Advisory Information and either Memory Mapped Files or Shared Memory Objects</b>	
307		The functionality described is optional. The functionality described is also an extension to the	
308		ISO C standard.	
309		This is a shorthand notation for combinations of multiple option codes.	
310		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.	
311		Where additional semantics apply to a function, the material is identified by use of the MC1	
312		margin legend.	
313		Refer to Section 1.5.2 (on page 14).	
314	MC2	<b>Memory Mapped Files, Shared Memory Objects, or Memory Protection</b>	
315		The functionality described is optional. The functionality described is also an extension to the	
316		ISO C standard.	
317		This is a shorthand notation for combinations of multiple option codes.	
318		Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.	
319		Where additional semantics apply to a function, the material is identified by use of the MC2	
320		margin legend.	
321		Refer to Section 1.5.2 (on page 14).	
322	MC3	<b>Memory Mapped Files, Shared Memory Objects, or Typed Memory Objects</b>	1
323		The functionality described is optional. The functionality described is also an extension to the	1
324		ISO C standard.	1
325		This is a shorthand notation for combinations of multiple option codes.	1
326		Where applicable, functions are marked with the MC3 margin legend in the SYNOPSIS section.	1
327		Where additional semantics apply to a function, the material is identified by use of the MC3	1
328		margin legend.	1
329		Refer to Section 1.5.2 (on page 14).	1
330	MF	<b>Memory Mapped Files</b>	
331		The functionality described is optional. The functionality described is also an extension to the	
332		ISO C standard.	

333		Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
334		Where additional semantics apply to a function, the material is identified by use of the MF
335		margin legend.
336	ML	<b>Process Memory Locking</b>
337		The functionality described is optional. The functionality described is also an extension to the
338		ISO C standard.
339		Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
340		Where additional semantics apply to a function, the material is identified by use of the ML
341		margin legend.
342	MLR	<b>Range Memory Locking</b>
343		The functionality described is optional. The functionality described is also an extension to the
344		ISO C standard.
345		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
346		Where additional semantics apply to a function, the material is identified by use of the MLR
347		margin legend.
348	MON	<b>Monotonic Clock</b>
349		The functionality described is optional. The functionality described is also an extension to the
350		ISO C standard.
351		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
352		Where additional semantics apply to a function, the material is identified by use of the MON
353		margin legend.
354	MPR	<b>Memory Protection</b>
355		The functionality described is optional. The functionality described is also an extension to the
356		ISO C standard.
357		Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
358		Where additional semantics apply to a function, the material is identified by use of the MPR
359		margin legend.
360	MSG	<b>Message Passing</b>
361		The functionality described is optional. The functionality described is also an extension to the
362		ISO C standard.
363		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
364		Where additional semantics apply to a function, the material is identified by use of the MSG
365		margin legend.
366	MX	<b>IEC 60559 Floating-Point Option</b>
367		The functionality described is optional. The functionality described is also an extension to the
368		ISO C standard.
369		Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
370		Where additional semantics apply to a function, the material is identified by use of the MX
371		margin legend.
372	OB	<b>Obsolescent</b>
373		The functionality described may be withdrawn in a future version of this volume of
374		IEEE Std 1003.1-2001. Strictly Conforming POSIX Applications and Strictly Conforming XSI
375		Applications shall not use obsolescent features.
376		Where applicable, the material is identified by use of the OB margin legend.

377 OF **Output Format Incompletely Specified**

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

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

382 OH **Optional Header**

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

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

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

390 PIO **Prioritized Input and Output**

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

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

396 PS **Process Scheduling**

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

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

402 RS **Raw Sockets**

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

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

408 RTS **Realtime Signals Extension**

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

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

414 SD **Software Development Utilities**

415 The functionality described is optional.

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

419 SEM **Semaphores**

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



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

#### 425 SHM Shared Memory Objects

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

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

#### 431 SIO Synchronized Input and Output

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

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

#### 437 SPI Spin Locks

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

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

#### 443 SPN Spawn

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

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

#### 449 SS Process Sporadic Server

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

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

#### 455 TCT Thread CPU-Time Clocks

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

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

#### 461 TEF Trace Event Filter

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

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

467	THR	<b>Threads</b>
468		The functionality described is optional. The functionality described is also an extension to the
469		ISO C standard.
470		Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
471		Where additional semantics apply to a function, the material is identified by use of the THR
472		margin legend.
473	TMO	<b>Timeouts</b>
474		The functionality described is optional. The functionality described is also an extension to the
475		ISO C standard.
476		Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
477		Where additional semantics apply to a function, the material is identified by use of the TMO
478		margin legend.
479	TMR	<b>Timers</b>
480		The functionality described is optional. The functionality described is also an extension to the
481		ISO C standard.
482		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
483		Where additional semantics apply to a function, the material is identified by use of the TMR
484		margin legend.
485	TPI	<b>Thread Priority Inheritance</b>
486		The functionality described is optional. The functionality described is also an extension to the
487		ISO C standard.
488		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
489		Where additional semantics apply to a function, the material is identified by use of the TPI
490		margin legend.
491	TPP	<b>Thread Priority Protection</b>
492		The functionality described is optional. The functionality described is also an extension to the
493		ISO C standard.
494		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
495		Where additional semantics apply to a function, the material is identified by use of the TPP
496		margin legend.
497	TPS	<b>Thread Execution Scheduling</b>
498		The functionality described is optional. The functionality described is also an extension to the
499		ISO C standard.
500		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
501		Where additional semantics apply to a function, the material is identified by use of the TPS
502		margin legend.
503	TRC	<b>Trace</b>
504		The functionality described is optional. The functionality described is also an extension to the
505		ISO C standard.
506		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
507		Where additional semantics apply to a function, the material is identified by use of the TRC
508		margin legend.
509	TRI	<b>Trace Inherit</b>
510		The functionality described is optional. The functionality described is also an extension to the
511		ISO C standard.

512		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
513		Where additional semantics apply to a function, the material is identified by use of the TRI
514		margin legend.
515	TRL	<b>Trace Log</b>
516		The functionality described is optional. The functionality described is also an extension to the
517		ISO C standard.
518		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
519		Where additional semantics apply to a function, the material is identified by use of the TRL
520		margin legend.
521	TSA	<b>Thread Stack Address Attribute</b>
522		The functionality described is optional. The functionality described is also an extension to the
523		ISO C standard.
524		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
525		Where additional semantics apply to a function, the material is identified by use of the TSA
526		margin legend.
527	TSF	<b>Thread-Safe Functions</b>
528		The functionality described is optional. The functionality described is also an extension to the
529		ISO C standard.
530		Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
531		Where additional semantics apply to a function, the material is identified by use of the TSF
532		margin legend.
533	TSH	<b>Thread Process-Shared Synchronization</b>
534		The functionality described is optional. The functionality described is also an extension to the
535		ISO C standard.
536		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
537		Where additional semantics apply to a function, the material is identified by use of the TSH
538		margin legend.
539	TSP	<b>Thread Sporadic Server</b>
540		The functionality described is optional. The functionality described is also an extension to the
541		ISO C standard.
542		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
543		Where additional semantics apply to a function, the material is identified by use of the TSP
544		margin legend.
545	TSS	<b>Thread Stack Size Attribute</b>
546		The functionality described is optional. The functionality described is also an extension to the
547		ISO C standard.
548		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
549		Where additional semantics apply to a function, the material is identified by use of the TSS
550		margin legend.
551	TYM	<b>Typed Memory Objects</b>
552		The functionality described is optional. The functionality described is also an extension to the
553		ISO C standard.
554		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
555		Where additional semantics apply to a function, the material is identified by use of the TYM
556		margin legend.

557	UP	<b>User Portability Utilities</b>
558		The functionality described is optional.
559		Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
560		Where additional semantics apply to a utility, the material is identified by use of the UP margin
561		legend.
562	XSI	<b>Extension</b>
563		The functionality described is an XSI extension. Functionality marked XSI is also an extension to
564		the ISO C standard. Application writers may confidently make use of an extension on all
565		systems supporting the X/Open System Interfaces Extension.
566		If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that
567		reference page is an extension. See Section 2.1.4 (on page 21).
568	XSR	<b>XSI STREAMS</b>
569		The functionality described is optional. The functionality described is also an extension to the
570		ISO C standard.
571		Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.
572		Where additional semantics apply to a function, the material is identified by use of the XSR
573		margin legend.
574	<b>1.5.2</b>	<b>Margin Code Notation</b>
575		Some of the functionality described in IEEE Std 1003.1-2001 depends on support of more than
576		one option, or independently may depend on several options. The following notation for margin
577		codes is used to denote the following cases.
578		<b>A Feature Dependent on One or Two Options</b>
579		In this case, margin codes have a <space> separator; for example:
580	MF	This feature requires support for only the Memory Mapped Files option.
581	MF SHM	This feature requires support for both the Memory Mapped Files and the Shared Memory
582		Objects options; that is, an application which uses this feature is portable only between
583		implementations that provide both options.
584		<b>A Feature Dependent on Either of the Options Denoted</b>
585		In this case, margin codes have a '   ' separator to denote the logical OR; for example:
586	MF SHM	This feature is dependent on support for either the Memory Mapped Files option or the Shared
587		Memory Objects option; that is, an application which uses this feature is portable between
588		implementations that provide any (or all) of the options.
589		<b>A Feature Dependent on More than Two Options</b>
590		The following shorthand notations are used:
591	MC1	The MC1 margin code is shorthand for ADV (MF SHM). Features which are shaded with this
592		margin code require support of the Advisory Information option and either the Memory
593		Mapped Files or Shared Memory Objects option.
594	MC2	The MC2 margin code is shorthand for MF SHM MPR. Features which are shaded with this
595		margin code require support of either the Memory Mapped Files, Shared Memory Objects, or
596		Memory Protection options.

597 MC3 The MC3 margin code is shorthand for MF|SHM|TYM. Features which are shaded with this 1  
598 margin code require support of either the Memory Mapped Files, Shared Memory Objects, or 1  
599 Typed Memory Objects options. 1

600 **Large Sections Dependent on an Option**

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

603 TRC This section describes extensions to support tracing of user applications. This functionality is  
604 dependent on support of the Trace option (and the rest of this section is not further shaded for  
605 this option).



## 2.1 Implementation Conformance

### 2.1.1 Requirements

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

1. The system shall support all utilities, functions, and facilities defined within IEEE Std 1003.1-2001 that are required for POSIX conformance (see Section 2.1.3 (on page 18)). These interfaces shall support the functional behavior described herein.
2. The system may support one or more options as described under Section 2.1.5 (on page 22). When an implementation claims that an option is supported, all of its constituent parts shall be provided.
3. The system may support the X/Open System Interface Extension (XSI) as described under Section 2.1.4 (on page 21).
4. The system may provide additional utilities, functions, or facilities not required by IEEE Std 1003.1-2001. Non-standard extensions of the utilities, functions, or facilities specified in IEEE Std 1003.1-2001 should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by IEEE Std 1003.1-2001. The conformance document shall define an environment in which an application can be run with the behavior specified by IEEE Std 1003.1-2001. In no case shall such an environment require modification of a Strictly Conforming POSIX Application (see Section 2.2.1 (on page 31)).

### 2.1.2 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to IEEE Std 1003.1-2001. The conformance document shall have the same structure as IEEE Std 1003.1-2001, with the information presented in the appropriate sections and subsections. Sections and subsections that consist solely of subordinate section titles, with no other information, are not required. The conformance document shall not contain information about extended facilities or capabilities outside the scope of IEEE Std 1003.1-2001.

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

The conformance document shall describe the limit values found in the headers `<limits.h>` (on page 249) and `<unistd.h>` (on page 403), stating values, the conditions under which those values may change, and the limits of such variations, if any.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in IEEE Std 1003.1-2001. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. When the value or behavior in the

implementation is designed to be variable or customized on each instantiation of the system, the implementation provider shall document the nature and permissible ranges of this variation.

The conformance document may specify the behavior of the implementation for those features where IEEE Std 1003.1-2001 states that implementations may vary or where features are identified as undefined or unspecified.

The conformance document shall not contain documentation other than that specified in the preceding paragraphs except where such documentation is specifically allowed or required by other provisions of IEEE Std 1003.1-2001.

The phrases “shall document” or “shall be documented” in IEEE Std 1003.1-2001 mean that documentation of the feature shall appear in the conformance document, as described previously, unless there is an explicit reference in the conformance document to show where the information can be found in the system documentation.

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

### 2.1.3 POSIX Conformance

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

#### 2.1.3.1 POSIX System Interfaces

- The system shall support all the mandatory functions and headers defined in IEEE Std 1003.1-2001, and shall set the symbolic constant `_POSIX_VERSION` to the value 200112L.
- Although all implementations conforming to IEEE Std 1003.1-2001 support all the features described below, there may be system-dependent or file system-dependent configuration procedures that can remove or modify any or all of these features. Such configurations should not be made if strict compliance is required.

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

— `_POSIX_CHOWN_RESTRICTED`

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

— `_POSIX_NO_TRUNC`

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

- The following symbolic constants shall be defined as follows:

- `_POSIX_JOB_CONTROL` shall have a value greater than zero.
- `_POSIX_SAVED_IDS` shall have a value greater than zero.
- `_POSIX_VDISABLE` shall have a value other than `-1`.

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



- 686 • The system may support one or more options (see Section 2.1.6 (on page 28)) denoted by the
- 687 following symbolic constants:
- 688 — \_POSIX\_ADVISORY\_INFO
- 689 — \_POSIX\_ASYNCHRONOUS\_IO
- 690 — \_POSIX\_BARRIERS
- 691 — \_POSIX\_CLOCK\_SELECTION
- 692 — \_POSIX\_CPUTIME
- 693 — \_POSIX\_FSYNC
- 694 — \_POSIX\_IPV6
- 695 — \_POSIX\_MAPPED\_FILES
- 696 — \_POSIX\_MEMLOCK
- 697 — \_POSIX\_MEMLOCK\_RANGE
- 698 — \_POSIX\_MEMORY\_PROTECTION
- 699 — \_POSIX\_MESSAGE\_PASSING
- 700 — \_POSIX\_MONOTONIC\_CLOCK
- 701 — \_POSIX\_PRIORITIZED\_IO
- 702 — \_POSIX\_PRIORITY\_SCHEDULING
- 703 — \_POSIX\_RAW\_SOCKETS
- 704 — \_POSIX\_REALTIME\_SIGNALS
- 705 — \_POSIX\_SEMAPHORES
- 706 — \_POSIX\_SHARED\_MEMORY\_OBJECTS
- 707 — \_POSIX\_SPAWN
- 708 — \_POSIX\_SPIN\_LOCKS
- 709 — \_POSIX\_SPORADIC\_SERVER
- 710 — \_POSIX\_SYNCHRONIZED\_IO
- 711 — \_POSIX\_THREAD\_ATTR\_STACKADDR
- 712 — \_POSIX\_THREAD\_CPUTIME
- 713 — \_POSIX\_THREAD\_ATTR\_STACKSIZE
- 714 — \_POSIX\_THREAD\_PRIO\_INHERIT
- 715 — \_POSIX\_THREAD\_PRIO\_PROTECT
- 716 — \_POSIX\_THREAD\_PRIORITY\_SCHEDULING
- 717 — \_POSIX\_THREAD\_PROCESS\_SHARED
- 718 — \_POSIX\_THREAD\_SAFE\_FUNCTIONS
- 719 — \_POSIX\_THREAD\_SPORADIC\_SERVER
- 720 — \_POSIX\_THREADS

721 — \_POSIX\_TIMEOUTS  
 722 — \_POSIX\_TIMERS  
 723 — \_POSIX\_TRACE  
 724 — \_POSIX\_TRACE\_EVENT\_FILTER  
 725 — \_POSIX\_TRACE\_INHERIT  
 726 — \_POSIX\_TRACE\_LOG  
 727 — \_POSIX\_TYPED\_MEMORY\_OBJECTS

728 If any of the symbolic constants \_POSIX\_TRACE\_EVENT\_FILTER, \_POSIX\_TRACE\_LOG, or  
 729 \_POSIX\_TRACE\_INHERIT is defined to have a value other than -1, then the symbolic  
 730 constant \_POSIX\_TRACE shall also be defined to have a value other than -1.

731 XSI • The system may support the XSI extensions (see Section 2.1.5.2 (on page 24)) denoted by the  
 732 following symbolic constants:

733 — \_XOPEN\_CRYPT  
 734 — \_XOPEN\_LEGACY  
 735 — \_XOPEN\_REALTIME  
 736 — \_XOPEN\_REALTIME\_THREADS  
 737 — \_XOPEN\_UNIX

### 738 2.1.3.2 POSIX Shell and Utilities

739 • The system shall provide all the mandatory utilities in the Shell and Utilities volume of  
 740 IEEE Std 1003.1-2001 with all the functional behavior described therein.

741 • The system shall support the Large File capabilities described in the Shell and Utilities  
 742 volume of IEEE Std 1003.1-2001.

743 • The system may support one or more options (see Section 2.1.6 (on page 28)) denoted by the  
 744 following symbolic constants. (The literal names below apply to the *getconf* utility.)

745 — POSIX2\_C\_DEV  
 746 — POSIX2\_CHAR\_TERM  
 747 — POSIX2\_FORT\_DEV  
 748 — POSIX2\_FORT\_RUN  
 749 — POSIX2\_LOCALEDEF  
 750 — POSIX2\_PBS  
 751 — POSIX2\_PBS\_ACCOUNTING  
 752 — POSIX2\_PBS\_LOCATE  
 753 — POSIX2\_PBS\_MESSAGE  
 754 — POSIX2\_PBS\_TRACK  
 755 — POSIX2\_SW\_DEV  
 756 — POSIX2\_UPE

- The system may support the XSI extensions (see Section 2.1.4).

Additional language bindings and development utility options may be provided in other related standards or in a future version of IEEE Std 1003.1-2001. In the former case, additional symbolic constants of the same general form as shown in this subsection should be defined by the related standard document and made available to the application without requiring IEEE Std 1003.1-2001 to be updated.

## 2.1.4 XSI Conformance

XSI This section describes the criteria for implementations conforming to the XSI extension (see Section 3.439 (on page 95)). This functionality is dependent on the support of the XSI extension (and the rest of this section is not further shaded).

IEEE Std 1003.1-2001 describes utilities, functions, and facilities offered to application programs by the X/Open System Interface (XSI). An XSI-conforming implementation shall meet the criteria for POSIX conformance and the following requirements.

### 2.1.4.1 XSI System Interfaces

- The system shall support all the functions and headers defined in IEEE Std 1003.1-2001 as part of the XSI extension denoted by the symbolic constant `_XOPEN_UNIX` and any extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)).
- The system shall support the `mmap()`, `munmap()`, and `msync()` functions.
- The system shall support the following options defined within IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)):
  - `_POSIX_FSYNC`
  - `_POSIX_MAPPED_FILES`
  - `_POSIX_MEMORY_PROTECTION`
  - `_POSIX_THREAD_ATTR_STACKADDR`
  - `_POSIX_THREAD_ATTR_STACKSIZE`
  - `_POSIX_THREAD_PROCESS_SHARED`
  - `_POSIX_THREAD_SAFE_FUNCTIONS`
  - `_POSIX_THREADS`
- The system may support the following XSI Option Groups (see Section 2.1.5.2 (on page 24)) defined within IEEE Std 1003.1-2001:
  - Encryption
  - Realtime
  - Advanced Realtime
  - Realtime Threads
  - Advanced Realtime Threads
  - Tracing
  - XSI STREAMS
  - Legacy

### 2.1.4.2 XSI Shell and Utilities Conformance

- The system shall support all the utilities defined in the Shell and Utilities volume of IEEE Std 1003.1-2001 as part of the XSI extension denoted by the XSI marking in the SYNOPSIS section, and any extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)) within the text.
  - The system shall support the User Portability Utilities option.
  - The system shall support creation of locales (see Chapter 7 (on page 123)).
  - The C-language Development utility *c99* shall be supported.
  - The XSI Development Utilities option may be supported. It consists of the following software development utilities:
- |              |              |               |              |   |
|--------------|--------------|---------------|--------------|---|
| <i>admin</i> | <i>delta</i> | <i>prs</i>    | <i>unget</i> | 1 |
| <i>cflow</i> | <i>get</i>   | <i>rm del</i> | <i>val</i>   | 1 |
| <i>ctags</i> | <i>m4</i>    | <i>sact</i>   | <i>what</i>  |   |
| <i>cxref</i> | <i>nm</i>    | <i>sccs</i>   |              | 1 |
- Within the utilities that are provided, functionality marked by the code OF (see Section 1.5.1 (on page 6)) need not be provided.

### 2.1.5 Option Groups

An Option Group is a group of related functions or options defined within the System Interfaces volume of IEEE Std 1003.1-2001.

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

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

#### 2.1.5.1 Subprofiling Considerations

Profiling standards supporting functional requirements less than that required in IEEE Std 1003.1-2001 may subset both mandatory and optional functionality required for POSIX Conformance (see Section 2.1.3 (on page 18)) or XSI Conformance (see Section 2.1.4 (on page 21)). Such profiles shall organize the subsets into Subprofiling Option Groups.

The Rationale (Informative) volume of IEEE Std 1003.1-2001, Appendix E, Subprofiling Considerations (Informative) describes a representative set of such Subprofiling Option Groups for use by profiles applicable to specialized realtime systems. IEEE Std 1003.1-2001 does not require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols defined in any header) or at runtime (via *sysconf()* or *getconf()*).

A Subprofiling Option Group may provide basic system functionality that other Subprofiling Option Groups and other options depend upon.<sup>3</sup> If a profile of IEEE Std 1003.1-2001 does not

3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are needed by any interface in IEEE Std 1003.1-2001 that parses a *path* argument. If a profile requires support for the Device Input and Output profiling option group but does not require support for the File System profiling option group, the profile must specify how pathname resolution is to behave in that profile, how the *O\_CREAT* flag to *open()* is to be handled (and the use of the character 'a' in the *mode* argument of *fopen()* when a filename argument names a file that does not exist), and specify lots of other details.

require an implementation to provide a Subprofiling Option Group that provides features utilized by a required Subprofiling Option Group (or option),<sup>4</sup> the profile shall specify<sup>5</sup> all of the following:

- Restricted or altered behavior of interfaces defined in IEEE Std 1003.1-2001 that may differ on an implementation of the profile
- Additional behaviors that may produce undefined or unspecified results
- Additional implementation-defined behavior that implementations shall be required to document in the profile's conformance document

if any of the above is a result of the profile not requiring an interface required by IEEE Std 1003.1-2001.

The following additional rules shall apply to all profiles of IEEE Std 1003.1-2001:

- Any application that conforms to that profile shall also conform to IEEE Std 1003.1-2001 (that is, a profile shall not require restricted, altered, or extended behaviors of an implementation of IEEE Std 1003.1-2001).
- Profiles are permitted to add additional requirements to the limits defined in **<limits.h>** and **<stdint.h>**, subject to the following:
  - For the limits in **<limits.h>** and **<stdint.h>**:
    - If the limit is specified as having a fixed value, it shall not be changed by a profile.
    - If a limit is specified as having a minimum or maximum acceptable value, it may be changed by a profile as follows:
      - A profile may increase a minimum acceptable value, but shall not make a minimum acceptable value smaller.
      - A profile may reduce a maximum acceptable value, but shall not make a maximum acceptable value larger.
  - A profile shall not change a limit specified as having a minimum or maximum value into a limit specified as having a fixed value.
  - A profile shall not create new limits.
  - Any implementation that conforms to IEEE Std 1003.1-2001 (including all options and extended limits required by the profile) shall also conform to that profile.

---

4. As an example, IEEE Std 1003.1-2001 requires that implementations claiming to support the Range Memory Locking option also support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application writer or system implementor would have to know to build an application or implementation conforming to the profile.

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

## 873 2.1.5.2 XSI Option Groups

874 XSI This section describes Option Groups to support the definition of XSI conformance within the  
 875 System Interfaces volume of IEEE Std 1003.1-2001. This functionality is dependent on the  
 876 support of the XSI extension (and the rest of this section is not further shaded).

877 The following Option Groups are defined.

878 **Encryption**

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

881 `crypt()`, `encrypt()`, `setkey()`

882 These functions are marked CRYPT.

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

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

889 **Realtime**

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

891 This Option Group includes a set of realtime functions drawn from options within  
 892 IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)).

893 Where entire functions are included in the Option Group, the NAME section is marked with  
 894 REALTIME. Where additional semantics have been added to existing pages, the new material is  
 895 identified by use of the appropriate margin legend for the underlying option defined within  
 896 IEEE Std 1003.1-2001.

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

899 This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001  
 900 (see Section 2.1.6 (on page 28)):

901 `_POSIX_ASYNCHRONOUS_IO`  
 902 `_POSIX_FSYNC`  
 903 `_POSIX_MAPPED_FILES`  
 904 `_POSIX_MEMLOCK`  
 905 `_POSIX_MEMLOCK_RANGE`  
 906 `_POSIX_MEMORY_PROTECTION`  
 907 `_POSIX_MESSAGE_PASSING`  
 908 `_POSIX_PRIORITIZED_IO`  
 909 `_POSIX_PRIORITY_SCHEDULING`  
 910 `_POSIX_REALTIME_SIGNALS`  
 911 `_POSIX_SEMAPHORES`  
 912 `_POSIX_SHARED_MEMORY_OBJECTS`  
 913 `_POSIX_SYNCHRONIZED_IO`  
 914 `_POSIX_TIMERS`

If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the following symbolic constants shall be defined by the implementation to have the value 200112L:

```

_POSIX_ASYNCHRONOUS_IO
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MESSAGE_PASSING
_POSIX_PRIORITY_SCHEDULING
_POSIX_REALTIME_SIGNALS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO
_POSIX_TIMERS

```

The functionality associated with `_POSIX_MAPPED_FILES`, `_POSIX_MEMORY_PROTECTION`, and `_POSIX_FSYNC` is always supported on XSI-conformant systems.

Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If this functionality is supported, then `_POSIX_PRIORITIZED_IO` shall be set to a value other than `-1`. Otherwise, it shall be undefined.

If `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling priority equal to a base scheduling priority minus `aiochp->aio_reqprio`. If Thread Execution Scheduling is not supported, then the base scheduling priority is that of the calling process; otherwise, the base scheduling priority is that of the calling thread. The implementation shall also document for which files I/O prioritization is supported.

### Advanced Realtime

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

Where entire functions are included in the Option Group, the NAME section is marked with ADVANCED REALTIME. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std 1003.1-2001.

This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)):

```

_POSIX_ADVISORY_INFO
_POSIX_CLOCK_SELECTION
_POSIX_CPUTIME
_POSIX_MONOTONIC_CLOCK
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TYPED_MEMORY_OBJECTS

```

If the implementation supports the Advanced Realtime Option Group, then the following symbolic constants shall be defined by the implementation to have the value 200112L:

957        \_POSIX\_ADVISORY\_INFO  
 958        \_POSIX\_CLOCK\_SELECTION  
 959        \_POSIX\_CPUTIME  
 960        \_POSIX\_MONOTONIC\_CLOCK  
 961        \_POSIX\_SPAWN  
 962        \_POSIX\_SPORADIC\_SERVER  
 963        \_POSIX\_TIMEOUTS  
 964        \_POSIX\_TYPED\_MEMORY\_OBJECTS

965        If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant  
 966        `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the  
 967        value 200112L.

968        If the symbolic constant `_POSIX_CPUTIME` is defined, then the symbolic constant  
 969        `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200112L.

970        If the symbolic constant `_POSIX_MONOTONIC_CLOCK` is defined, then the symbolic constant  
 971        `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200112L.

972        If the symbolic constant `_POSIX_CLOCK_SELECTION` is defined, then the symbolic constant  
 973        `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200112L.

#### 974        **Realtime Threads**

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

977        This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001  
 978        (see Section 2.1.6 (on page 28)):

979        \_POSIX\_THREAD\_PRIO\_INHERIT  
 980        \_POSIX\_THREAD\_PRIO\_PROTECT  
 981        \_POSIX\_THREAD\_PRIORITY\_SCHEDULING

982        Where applicable, whole pages are marked `REALTIME THREADS`, together with the  
 983        appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)).

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

986        If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the  
 987        following options shall also be defined by the implementation to have the value 200112L:

988        \_POSIX\_THREAD\_PRIO\_INHERIT  
 989        \_POSIX\_THREAD\_PRIO\_PROTECT  
 990        \_POSIX\_THREAD\_PRIORITY\_SCHEDULING

#### 991        **Advanced Realtime Threads**

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

994        Where entire functions are included in the Option Group, the NAME section is marked with  
 995        `ADVANCED REALTIME THREADS`. Where additional semantics have been added to existing  
 996        pages, the new material is identified by use of the appropriate margin legend for the underlying  
 997        option defined within IEEE Std 1003.1-2001.

998        This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001  
 999        (see Section 2.1.6 (on page 28)):



1000        \_POSIX\_BARRIERS  
 1001        \_POSIX\_SPIN\_LOCKS  
 1002        \_POSIX\_THREAD\_CPUTIME  
 1003        \_POSIX\_THREAD\_SPORADIC\_SERVER

1004        If the symbolic constant \_POSIX\_THREAD\_SPORADIC\_SERVER is defined to have the value  
 1005        200112L, then the symbolic constant \_POSIX\_THREAD\_PRIORITY\_SCHEDULING shall also be  
 1006        defined by the implementation to have the value 200112L.

1007        If the symbolic constant \_POSIX\_THREAD\_CPUTIME is defined to have the value 200112L,  
 1008        then the symbolic constant \_POSIX\_TIMERS shall also be defined by the implementation to have  
 1009        the value 200112L.

1010        If the symbolic constant \_POSIX\_BARRIERS is defined to have the value 200112L, then the  
 1011        symbolic constants \_POSIX\_THREADS and \_POSIX\_THREAD\_SAFE\_FUNCTIONS shall also  
 1012        be defined by the implementation to have the value 200112L.

1013        If the symbolic constant \_POSIX\_SPIN\_LOCKS is defined to have the value 200112L, then the  
 1014        symbolic constants \_POSIX\_THREADS and \_POSIX\_THREAD\_SAFE\_FUNCTIONS shall also  
 1015        be defined by the implementation to have the value 200112L.

1016        If the implementation supports the Advanced Realtime Threads Option Group, then the  
 1017        following symbolic constants shall be defined by the implementation to have the value 200112L:

1018        \_POSIX\_BARRIERS  
 1019        \_POSIX\_SPIN\_LOCKS  
 1020        \_POSIX\_THREAD\_CPUTIME  
 1021        \_POSIX\_THREAD\_SPORADIC\_SERVER

## 1022        **Tracing**

1023        This Option Group includes a set of tracing functions drawn from options within  
 1024        IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)).

1025        Where entire functions are included in the Option Group, the NAME section is marked with  
 1026        TRACING. Where additional semantics have been added to existing pages, the new material is  
 1027        identified by use of the appropriate margin legend for the underlying option defined within  
 1028        IEEE Std 1003.1-2001.

1029        This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001  
 1030        (see Section 2.1.6 (on page 28)):

1031        \_POSIX\_TRACE  
 1032        \_POSIX\_TRACE\_EVENT\_FILTER  
 1033        \_POSIX\_TRACE\_LOG  
 1034        \_POSIX\_TRACE\_INHERIT

1035        If the implementation supports the Tracing Option Group, then the following symbolic  
 1036        constants shall be defined by the implementation to have the value 200112L:

1037        \_POSIX\_TRACE  
 1038        \_POSIX\_TRACE\_EVENT\_FILTER  
 1039        \_POSIX\_TRACE\_LOG  
 1040        \_POSIX\_TRACE\_INHERIT

**XSI STREAMS**

The XSI STREAMS Option Group is denoted by the symbolic constant `_XOPEN_STREAMS`.

This Option Group includes functionality related to STREAMS, a uniform mechanism for implementing networking services and other character-based I/O as described in the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.6, STREAMS.

It includes the following functions:

*fattach()*, *fdetach()*, *getmsg()*, *getpmsg()*, *ioctl()*, *isastream()*, *putmsg()*, *putpmsg()*

and the `<stropts.h>` header.

Where applicable, whole pages are marked STREAMS, together with the appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)). Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std 1003.1-2001.

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

**Legacy**

The Legacy Option Group is denoted by the symbolic constant `_XOPEN_LEGACY`.

The Legacy Option Group includes the functions and headers which were mandatory in previous versions of IEEE Std 1003.1-2001 but are optional in this version.

These functions and headers are retained in IEEE Std 1003.1-2001 because of their widespread use. Application writers should not rely on the existence of these functions or headers in new applications, but should follow the migration path detailed in the APPLICATION USAGE sections of the relevant pages.

Various factors may have contributed to the decision to mark a function or header LEGACY. In all cases, the specific reasons for the withdrawal of a function or header are documented on the relevant pages.

Once a function or header is marked LEGACY, no modifications are made to the specifications of such functions or headers other than to the APPLICATION USAGE sections of the relevant pages.

The functions and headers which form this Option Group are as follows:

*bcmp()*, *bcopy()*, *bzero()*, *ecvt()*, *fcvt()*, *ftime()*, *gcvt()*, *getwd()*, *index()*, *mktemp()*, *rindex()*, *utimes()*, *wcs wcs()*

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

**2.1.6 Options**

The symbolic constants defined in `<unistd.h>`, **Constants for Options and Option Groups** (on page 403) reflect implementation options for IEEE Std 1003.1-2001. These symbols can be used by the application to determine which optional facilities are present on the implementation. The *sysconf()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 or the *getconf* utility defined in the Shell and Utilities volume of IEEE Std 1003.1-2001 can be used to retrieve the value of each symbol on each specific implementation to determine whether the option is supported.

1082		Where an option is not supported, the associated utilities, functions, or facilities need not be
1083		present.
1084		Margin codes are defined for each option (see Section 1.5.1 (on page 6)).
1085	2.1.6.1	<i>System Interfaces</i>
1086		Refer to < <b>unistd.h</b> >, <b>Constants for Options and Option Groups</b> (on page 403) for the list of
1087		options.
1088	2.1.6.2	<i>Shell and Utilities</i>
1089		Each of these symbols shall be considered valid names by the implementation. Refer to
1090		< <b>unistd.h</b> >, <b>Constants for Options and Option Groups</b> (on page 403).
1091		The literal names shown below apply only to the <i>getconf</i> utility.
1092	CD	POSIX2_C_DEV
1093		The system supports the C-Language Development Utilities option.
1094		The utilities in the C-Language Development Utilities option are used for the development
1095		of C-language applications, including compilation or translation of C source code and
1096		complex program generators for simple lexical tasks and processing of context-free
1097		grammars.
1098		The utilities listed below may be provided by a conforming system; however, any system
1099		claiming conformance to the C-Language Development Utilities option shall provide all of
1100		the utilities listed.
1101		<i>c99</i>
1102		<i>lex</i>
1103		<i>yacc</i>
1104		POSIX2_CHAR_TERM
1105		The system supports the Terminal Characteristics option. This value need not be present on
1106		a system not supporting the User Portability Utilities option.
1107		Where applicable, the dependency is noted within the description of the utility.
1108		This option applies only to systems supporting the User Portability Utilities option. If
1109		supported, then the system supports at least one terminal type capable of all operations
1110		described in IEEE Std 1003.1-2001; see Section 10.2 (on page 185).
1111	FD	POSIX2_FORT_DEV
1112		The system supports the FORTRAN Development Utilities option.
1113		The <i>fort77</i> FORTRAN compiler is the only utility in the FORTRAN Development Utilities
1114		option. This is used for the development of FORTRAN language applications, including
1115		compilation or translation of FORTRAN source code.
1116		The <i>fort77</i> utility may be provided by a conforming system; however, any system claiming
1117		conformance to the FORTRAN Development Utilities option shall provide the <i>fort77</i> utility.
1118	FR	POSIX2_FORT_RUN
1119		The system supports the FORTRAN Runtime Utilities option.
1120		The <i>asa</i> utility is the only utility in the FORTRAN Runtime Utilities option.
1121		The <i>asa</i> utility may be provided by a conforming system; however, any system claiming
1122		conformance to the FORTRAN Runtime Utilities option shall provide the <i>asa</i> utility.

1123		POSIX2_LOCALEDEF
1124		The system supports the Locale Creation Utilities option.
1125		If supported, the system supports the creation of locales as described in the <i>localedef</i> utility.
1126		The <i>localedef</i> utility may be provided by a conforming system; however, any system
1127		claiming conformance to the Locale Creation Utilities option shall provide the <i>localedef</i>
1128		utility.
1129	BE	POSIX2_PBS
1130		The system supports the Batch Environment Services and Utilities option (see the Shell and
1131		Utilities volume of IEEE Std 1003.1-2001, Chapter 3, Batch Environment Services).
1132		<b>Note:</b> The Batch Environment Services and Utilities option is a combination of mandatory and
1133		optional batch services and utilities. The POSIX_PBS symbolic constant implies the
1134		system supports all the mandatory batch services and utilities.
1135		POSIX2_PBS_ACCOUNTING
1136		The system supports the Batch Accounting option.
1137		POSIX2_PBS_CHECKPOINT
1138		The system supports the Batch Checkpoint/Restart option.
1139		POSIX2_PBS_LOCATE
1140		The system supports the Locate Batch Job Request option.
1141		POSIX2_PBS_MESSAGE
1142		The system supports the Batch Job Message Request option.
1143		POSIX2_PBS_TRACK
1144		The system supports the Track Batch Job Request option.
1145	SD	POSIX2_SW_DEV
1146		The system supports the Software Development Utilities option.
1147		The utilities in the Software Development Utilities option are used for the development of
1148		applications, including compilation or translation of source code, the creation and
1149		maintenance of library archives, and the maintenance of groups of inter-dependent
1150		programs.
1151		The utilities listed below may be provided by the conforming system; however, any system
1152		claiming conformance to the Software Development Utilities option shall provide all of the
1153		utilities listed here.
1154		<i>ar</i>
1155		<i>make</i>
1156		<i>nm</i>
1157		<i>strip</i>
1158	UP	POSIX2_UPE
1159		The system supports the User Portability Utilities option.
1160		The utilities in the User Portability Utilities option shall be implemented on all systems that
1161		claim conformance to this option. Certain utilities are noted as having features that cannot
1162		be implemented on all terminal types; if the POSIX2_CHAR_TERM option is supported, the
1163		system shall support all such features on at least one terminal type; see Section 10.2 (on
1164		page 185).
1165		Some of the utilities are required only on systems that also support the Software
1166		Development Utilities option, or the character-at-a-time terminal option (see Section 10.2
1167		(on page 185)); such utilities have this noted in their DESCRIPTION sections. All of the

1168 other utilities listed are required only on systems that claim conformance to the User  
 1169 Portability Utilities option.

1170	<i>alias</i>	<i>expand</i>	<i>nm</i>	<i>unalias</i>
1171	<i>at</i>	<i>fc</i>	<i>patch</i>	<i>unexpand</i>
1172	<i>batch</i>	<i>fg</i>	<i>ps</i>	<i>uudecode</i>
1173	<i>bg</i>	<i>file</i>	<i>renice</i>	<i>uuencode</i>
1174	<i>crontab</i>	<i>jobs</i>	<i>split</i>	<i>vi</i>
1175	<i>split</i>	<i>man</i>	<i>strings</i>	<i>who</i>
1176	<i>ctags</i>	<i>mesg</i>	<i>tabs</i>	<i>write</i>
1177	<i>df</i>	<i>more</i>	<i>talk</i>	
1178	<i>du</i>	<i>newgrp</i>	<i>time</i>	
1179	<i>ex</i>	<i>nice</i>	<i>tput</i>	

## 1180 2.2 Application Conformance

1181 All applications claiming conformance to IEEE Std 1003.1-2001 shall use only language-  
 1182 dependent services for the C programming language described in Section 2.3 (on page 33), shall  
 1183 use only the utilities and facilities defined in the Shell and Utilities volume of  
 1184 IEEE Std 1003.1-2001, and shall fall within one of the following categories.

### 1185 2.2.1 Strictly Conforming POSIX Application

1186 A Strictly Conforming POSIX Application is an application that requires only the facilities  
 1187 described in IEEE Std 1003.1-2001. Such an application:

- 1188 1. Shall accept any implementation behavior that results from actions it takes in areas  
 1189 described in IEEE Std 1003.1-2001 as *implementation-defined* or *unspecified*, or where  
 1190 IEEE Std 1003.1-2001 indicates that implementations may vary
- 1191 2. Shall not perform any actions that are described as producing *undefined* results
- 1192 3. For symbolic constants, shall accept any value in the range permitted by  
 1193 IEEE Std 1003.1-2001, but shall not rely on any value in the range being greater than the  
 1194 minimums listed or being less than the maximums listed in IEEE Std 1003.1-2001
- 1195 4. Shall not use facilities designated as *obsolescent*
- 1196 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
 1197 facilities whose availability is indicated by Section 2.1.3 (on page 18)
- 1198 6. For the C programming language, shall not produce any output dependent on any  
 1199 behavior described in the ISO/IEC 9899:1999 standard as *unspecified*, *undefined*, or  
 1200 *implementation-defined*, unless the System Interfaces volume of IEEE Std 1003.1-2001  
 1201 specifies the behavior
- 1202 7. For the C programming language, shall not exceed any minimum implementation limit  
 1203 defined in the ISO/IEC 9899:1999 standard, unless the System Interfaces volume of  
 1204 IEEE Std 1003.1-2001 specifies a higher minimum implementation limit
- 1205 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200112L before  
 1206 any header is included

1207 Within IEEE Std 1003.1-2001, any restrictions placed upon a Conforming POSIX Application  
 1208 shall restrict a Strictly Conforming POSIX Application.

## 1209 2.2.2 Conforming POSIX Application

### 1210 2.2.2.1 ISO/IEC Conforming POSIX Application

1211 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities  
1212 described in IEEE Std 1003.1-2001 and approved Conforming Language bindings for any ISO or  
1213 IEC standard. Such an application shall include a statement of conformance that documents all  
1214 options and limit dependencies, and all other ISO or IEC standards used.

### 1215 2.2.2.2 <National Body> Conforming POSIX Application

1216 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming  
1217 POSIX Application in that it also may use specific standards of a single ISO/IEC member body  
1218 referred to here as <National Body>. Such an application shall include a statement of  
1219 conformance that documents all options and limit dependencies, and all other <National Body>  
1220 standards used.

## 1221 2.2.3 Conforming POSIX Application Using Extensions

1222 A Conforming POSIX Application Using Extensions is an application that differs from a  
1223 Conforming POSIX Application only in that it uses non-standard facilities that are consistent  
1224 with IEEE Std 1003.1-2001. Such an application shall fully document its requirements for these  
1225 extended facilities, in addition to the documentation required of a Conforming POSIX  
1226 Application. A Conforming POSIX Application Using Extensions shall be either an ISO/IEC  
1227 Conforming POSIX Application Using Extensions or a <National Body> Conforming POSIX  
1228 Application Using Extensions (see Section 2.2.2.1 and Section 2.2.2.2).

## 1229 2.2.4 Strictly Conforming XSI Application

1230 A Strictly Conforming XSI Application is an application that requires only the facilities described  
1231 in IEEE Std 1003.1-2001. Such an application:

- 1232 1. Shall accept any implementation behavior that results from actions it takes in areas  
1233 described in IEEE Std 1003.1-2001 as *implementation-defined* or *unspecified*, or where  
1234 IEEE Std 1003.1-2001 indicates that implementations may vary
- 1235 2. Shall not perform any actions that are described as producing *undefined* results
- 1236 3. For symbolic constants, shall accept any value in the range permitted by  
1237 IEEE Std 1003.1-2001, but shall not rely on any value in the range being greater than the  
1238 minimums listed or being less than the maximums listed in IEEE Std 1003.1-2001
- 1239 4. Shall not use facilities designated as *obsolescent*
- 1240 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
1241 facilities whose availability is indicated by Section 2.1.4 (on page 21)
- 1242 6. For the C programming language, shall not produce any output dependent on any  
1243 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*  
1244 *defined*, unless the System Interfaces volume of IEEE Std 1003.1-2001 specifies the behavior
- 1245 7. For the C programming language, shall not exceed any minimum implementation limit  
1246 defined in the ISO C standard, unless the System Interfaces volume of  
1247 IEEE Std 1003.1-2001 specifies a higher minimum implementation limit
- 1248 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 600 before any  
1249 header is included

1250 Within IEEE Std 1003.1-2001, any restrictions placed upon a Conforming POSIX Application  
1251 shall restrict a Strictly Conforming XSI Application.

#### 1252 **2.2.5 Conforming XSI Application Using Extensions**

1253 A Conforming XSI Application Using Extensions is an application that differs from a Strictly  
1254 Conforming XSI Application only in that it uses non-standard facilities that are consistent with  
1255 IEEE Std 1003.1-2001. Such an application shall fully document its requirements for these  
1256 extended facilities, in addition to the documentation required of a Strictly Conforming XSI  
1257 Application.

### 1258 **2.3 Language-Dependent Services for the C Programming Language**

1259 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX  
1260 conformance as described in Section 2.1.3 (on page 18).

### 1261 **2.4 Other Language-Related Specifications**

1262 IEEE Std 1003.1-2001 is currently specified in terms of the shell command language and ISO C.  
1263 Bindings to other programming languages are being developed.

1264 If conformance to IEEE Std 1003.1-2001 is claimed for implementation of any programming  
1265 language, the implementation of that language shall support the use of external symbols distinct  
1266 to at least 31 bytes in length in the source program text. (That is, identifiers that differ at or  
1267 before the thirty-first byte shall be distinct.) If a national or international standard governing a  
1268 language defines a maximum length that is less than this value, the language-defined maximum  
1269 shall be supported. External symbols that differ only by case shall be distinct when the character  
1270 set in use distinguishes uppercase and lowercase characters and the language permits (or  
1271 requires) uppercase and lowercase characters to be distinct in external symbols.





## Definitions

1273

1274 For the purposes of IEEE Std 1003.1-2001, the terms and definitions given in Chapter 3 apply.

1275 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and  
1276 definitions given in this chapter are used elsewhere in text related to extensions and options,  
1277 they are shaded as appropriate.

### 1278 3.1 Abortive Release

1279 An abrupt termination of a network connection that may result in the loss of data.

### 1280 3.2 Absolute Pathname

1281 A pathname beginning with a single or more than two slashes; see also Section 3.266 (on page  
1282 72).

1283 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 100).

### 1284 3.3 Access Mode

1285 A particular form of access permitted to a file.

### 1286 3.4 Additional File Access Control Mechanism

1287 An implementation-defined mechanism that is layered upon the access control mechanisms  
1288 defined here, but which do not grant permissions beyond those defined herein, although they  
1289 may further restrict them.

1290 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 97).

### 1291 3.5 Address Space

1292 The memory locations that can be referenced by a process or the threads of a process.

### 1293 3.6 Advisory Information

1294 An interface that advises the implementation on (portable) application behavior so that it can  
1295 optimize the system.

### 1296 3.7 Affirmative Response

1297 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category  
1298 keyword **yesexpr**, matching an extended regular expression in the current locale.

1299 **Note:** The *LC\_MESSAGES* category is defined in detail in Section 7.3.6 (on page 152).

### 1300 3.8 Alert

1301 To cause the user's terminal to give some audible or visual indication that an error or some other  
1302 event has occurred. When the standard output is directed to a terminal device, the method for  
1303 alerting the terminal user is unspecified. When the standard output is not directed to a terminal  
1304 device, the alert is accomplished by writing the <alert> to standard output (unless the utility  
1305 description indicates that the use of standard output produces undefined results in this case).

### 1306 3.9 Alert Character (<alert>)

1307 A character that in the output stream should cause a terminal to alert its user via a visual or  
1308 audible notification. It is the character designated by '`\a`' in the C language. It is unspecified  
1309 whether this character is the exact sequence transmitted to an output device by the system to  
1310 accomplish the alert function.

### 1311 3.10 Alias Name

1312 In the shell command language, a word consisting solely of underscores, digits, and alphabets  
1313 from the portable character set and any of the following characters: '`!`', '`%`', '`,`', '`'`', '`@`'.

1314 Implementations may allow other characters within alias names as an extension.

1315 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 113).

### 1316 3.11 Alignment

1317 A requirement that objects of a particular type be located on storage boundaries with addresses  
1318 that are particular multiples of a byte address.

1319 **Note:** See also the ISO C standard, Section B3.

### 1320 3.12 Alternate File Access Control Mechanism

1321 An implementation-defined mechanism that is independent of the access control mechanisms  
1322 defined herein, and which if enabled on a file may either restrict or extend the permissions of a  
1323 given user. IEEE Std 1003.1-2001 defines when such mechanisms can be enabled and when they  
1324 are disabled.

1325 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 97).

**1326 3.13 Alternate Signal Stack**

1327 Memory associated with a thread, established upon request by the implementation for a thread,  
1328 separate from the thread signal stack, in which signal handlers responding to signals sent to that  
1329 thread may be executed.

**1330 3.14 Ancillary Data**

1331 Protocol-specific, local system-specific, or optional information. The information can be both  
1332 local or end-to-end significant, header information, part of a data portion, protocol-specific, and  
1333 implementation or system-specific.

**1334 3.15 Angle Brackets**

1335 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase  
1336 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed,  
1337 and '`>`' immediately follows it. When describing these characters in the portable character set,  
1338 the names `<less-than-sign>` and `<greater-than-sign>` are used.

**1339 3.16 Application**

1340 A computer program that performs some desired function.

**1341 3.17 Application Address**

1342 Endpoint address of a specific application.

**1343 3.18 Application Program Interface (API)**

1344 The definition of syntax and semantics for providing computer system services.

**1345 3.19 Appropriate Privileges**

1346 An implementation-defined means of associating privileges with a process with regard to the  
1347 function calls, function call options, and the commands that need special privileges. There may  
1348 be zero or more such means. These means (or lack thereof) are described in the conformance  
1349 document.

1350 **Note:** Function calls are defined in the System Interfaces volume of IEEE Std 1003.1-2001, and  
1351 commands are defined in the Shell and Utilities volume of IEEE Std 1003.1-2001.

**1352 3.20 Argument**

1353 In the shell command language, a parameter passed to a utility as the equivalent of a single  
1354 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,  
1355 option-arguments, or operands following the command name.

1356 **Note:** The Utility Argument Syntax is defined in detail in Section 12.1 (on page 201) and the Shell and  
1357 Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

1358 In the C language, an expression in a function call expression or a sequence of preprocessing  
1359 tokens in a function-like macro invocation.

**1360 3.21 Arm (a Timer)**

1361 To start a timer measuring the passage of time, enabling notifying a process when the specified  
1362 time or time interval has passed.

**1363 3.22 Asterisk**

1364 The character ' \* '.

**1365 3.23 Async-Cancel-Safe Function**

1366 A function that may be safely invoked by an application while the asynchronous form of  
1367 cancellation is enabled. No function is async-cancel-safe unless explicitly described as such.

**1368 3.24 Asynchronous Events**

1369 Events that occur independently of the execution of the application.

**1370 3.25 Asynchronous Input and Output**

1371 A functionality enhancement to allow an application process to queue data input and output  
1372 commands with asynchronous notification of completion.

**1373 3.26 Async-Signal-Safe Function**

1374 A function that may be invoked, without restriction, from signal-catching functions. No function  
1375 is async-signal-safe unless explicitly described as such.

**1376 3.27 Asynchronously-Generated Signal**

1377 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals  
1378 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a  
1379 property of how the signal was generated and not a property of the signal number. All signals  
1380 may be generated asynchronously.

1381 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of IEEE Std 1003.1-2001.

**1382 3.28 Asynchronous I/O Completion**

1383 For an asynchronous read or write operation, when a corresponding synchronous read or write  
1384 would have completed and when any associated status fields have been updated.

**1385 3.29 Asynchronous I/O Operation**

1386 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from  
1387 further use of the processor.

1388 This implies that the process and the I/O operation may be running concurrently.

**1389 3.30 Authentication**

1390 The process of validating a user or process to verify that the user or process is not a counterfeit.

**1391 3.31 Authorization**

1392 The process of verifying that a user or process has permission to use a resource in the manner  
1393 requested.

1394 To ensure security, the user or process would also need to be authenticated before granting  
1395 access.

**1396 3.32 Background Job**

1397 See *Background Process Group* in Section 3.34.

**1398 3.33 Background Process**

1399 A process that is a member of a background process group.

**1400 3.34 Background Process Group (or Background Job)**

1401 Any process group, other than a foreground process group, that is a member of a session that  
1402 has established a connection with a controlling terminal.

**1403 3.35 Backquote**

1404 The character ' ` ', also known as a grave accent.

**1405 3.36 Backslash**

1406 The character ' \ ', also known as a reverse solidus.

**1407 3.37 Backspace Character (<backspace>)**

1408 A character that, in the output stream, should cause printing (or displaying) to occur one column  
1409 position previous to the position about to be printed. If the position about to be printed is at the  
1410 beginning of the current line, the behavior is unspecified. It is the character designated by ' \b '  
1411 in the C language. It is unspecified whether this character is the exact sequence transmitted to an  
1412 output device by the system to accomplish the backspace function. The <backspace> defined  
1413 here is not necessarily the ERASE special character.

1414 **Note:** Special Characters are defined in detail in Section 11.1.9 (on page 191).

**1415 3.38 Barrier**

1416 A synchronization object that allows multiple threads to synchronize at a particular point in  
1417 their execution.

**1418 3.39 Base Character**

1419 One of the set of characters defined in the Latin alphabet. In Western European languages other  
1420 than English, these characters are commonly used with diacritical marks (accents, cedilla, and so  
1421 on) to extend the range of characters in an alphabet.

**1422 3.40 Basename**

1423 The final, or only, filename in a pathname.

**1424 3.41 Basic Regular Expression (BRE)**

1425 A regular expression (see Section 3.316 (on page 79)) used by the majority of utilities that select  
1426 strings from a set of character strings.

1427 **Note:** Basic Regular Expressions are described in detail in Section 9.3 (on page 171).

**1428 3.42 Batch Access List**

1429 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a  
1430 batch queue.

1431 A batch access list is associated with a batch queue. A batch server uses the batch access list of a  
1432 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

**1433 3.43 Batch Administrator**

1434 A user that is authorized to modify all the attributes of queues and jobs and to change the status  
1435 of a batch server.

**1436 3.44 Batch Client**

1437 A computational entity that utilizes batch services by making requests of batch servers.  
1438 Batch clients often provide the means by which users access batch services, although a batch  
1439 server may act as a batch client by virtue of making requests of another batch server.

**1440 3.45 Batch Destination**

1441 The batch server in a batch system to which a batch job should be sent for processing.  
1442 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.  
1443 A batch destination may consist of a batch server-specific portion, a network-wide portion, or  
1444 both. The batch server-specific portion is referred to as the “batch queue”. The network-wide  
1445 portion is referred to as a “batch server name”.

**1446 3.46 Batch Destination Identifier**

1447 A string that identifies a specific batch destination.  
1448 A string of characters in the portable character set used to specify a particular batch destination.  
1449 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 113).

**1450 3.47 Batch Directive**

1451 A line from a file that is interpreted by the batch server. The line is usually in the form of a  
1452 comment and is an additional means of passing options to the *qsub* utility.  
1453 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

**1454 3.48 Batch Job**

1455 A set of computational tasks for a computing system.  
1456 Batch jobs are managed by batch servers.  
1457 Once created, a batch job may be executing or pending execution. A batch job that is executing  
1458 has an associated session leader (a process) that initiates and monitors the computational tasks  
1459 of the batch job.

**1460 3.49 Batch Job Attribute**

1461 A named data type whose value affects the processing of a batch job.

1462 The values of the attributes of a batch job affect the processing of that job by the batch server  
1463 that manages the batch job.

**1464 3.50 Batch Job Identifier**

1465 A unique name for a batch job. A name that is unique among all other batch job identifiers in a  
1466 batch system and that identifies the batch server to which the batch job was originally  
1467 submitted.

**1468 3.51 Batch Job Name**

1469 A label that is an attribute of a batch job. The batch job name is not necessarily unique.

**1470 3.52 Batch Job Owner**

1471 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see  
1472 also Section 3.426 (on page 93)) and *hostname* is a network host name.

**1473 3.53 Batch Job Priority**

1474 A value specified by the user that may be used by an implementation to determine the order in  
1475 which batch jobs are selected to be executed. Job priority has a numeric value in the range –1 024  
1476 to 1 023.

1477 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.

**1478 3.54 Batch Job State**

1479 An attribute of a batch job which determines the types of requests that the batch server that  
1480 manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING,  
1481 HELD, WAITING, EXITING, and TRANSITING.

**1482 3.55 Batch Name Service**

1483 A service that assigns batch names that are unique within the batch name space, and that can  
1484 translate a unique batch name into the location of the named batch entity.

**1485 3.56 Batch Name Space**

1486 The environment within which a batch name is known to be unique.



**1487 3.57 Batch Node**

1488 A host containing part or all of a batch system.

1489 A batch node is a host meeting at least one of the following conditions:

- 1490 • Capable of executing a batch client
- 1491 • Contains a routing batch queue
- 1492 • Contains an execution batch queue

**1493 3.58 Batch Operator**

1494 A user that is authorized to modify some, but not all, of the attributes of jobs and queues, and  
1495 may change the status of the batch server.

**1496 3.59 Batch Queue**

1497 A manageable object that represents a set of batch jobs and is managed by a single batch server.

1498 **Note:** A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from  
1499 the batch queue for execution based on attributes such as priority, resource requirements, and  
1500 hold conditions.

1501 See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 3.1.2, Batch Queues.

**1502 3.60 Batch Queue Attribute**

1503 A named data type whose value affects the processing of all batch jobs that are members of the  
1504 batch queue.

1505 A batch queue has attributes that affect the processing of batch jobs that are members of the  
1506 batch queue.

**1507 3.61 Batch Queue Position**

1508 The place, relative to other jobs in the batch queue, occupied by a particular job in a batch queue.  
1509 This is defined in part by submission time and priority; see also Section 3.62.

**1510 3.62 Batch Queue Priority**

1511 The maximum job priority allowed for any batch job in a given batch queue.

1512 The batch queue priority is set and may be changed by users with appropriate privilege. The  
1513 priority is bounded in an implementation-defined manner.

**1514 3.63 Batch Rerunability**

1515 An attribute of a batch job indicating that it may be rerun after an abnormal termination from  
1516 the beginning without affecting the validity of the results.

**1517 3.64 Batch Restart**

1518 The action of resuming the processing of a batch job from the point of the last checkpoint.  
1519 Typically, this is done if the batch job has been interrupted because of a system failure.

**1520 3.65 Batch Server**

1521 A computational entity that provides batch services.

**1522 3.66 Batch Server Name**

1523 A string of characters in the portable character set used to specify a particular server in a  
1524 network.

1525 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 113).

**1526 3.67 Batch Service**

1527 Computational and organizational services performed by a batch system on behalf of batch jobs.

1528 Batch services are of two types: requested and deferred.

1529 **Note:** Batch Services are listed in the Shell and Utilities volume of IEEE Std 1003.1-2001, Table 3-5,  
1530 Batch Services Summary.

**1531 3.68 Batch Service Request**

1532 A solicitation of services from a batch client to a batch server.

1533 A batch service request may entail the exchange of any number of messages between the batch  
1534 client and the batch server.

1535 When naming specific types of service requests, the term “request” is qualified by the type of  
1536 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

**1537 3.69 Batch Submission**

1538 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*  
1539 *Request* to perform a specified computational task.

**1540 3.70 Batch System**

1541 A collection of one or more batch servers.

### 1542 **3.71 Batch Target User**

1543 The name of a user on the batch destination batch server.

1544 The target user is the user name under whose account the batch job is to execute on the  
1545 destination batch server.

### 1546 **3.72 Batch User**

1547 A user who is authorized to make use of batch services.

### 1548 **3.73 Bind**

1549 The process of assigning a network address to an endpoint.

### 1550 **3.74 Blank Character (<blank>)**

1551 One of the characters that belong to the **blank** character class as defined via the *LC\_CTYPE*  
1552 category in the current locale. In the POSIX locale, a <blank> is either a <tab> or a <space>.

### 1553 **3.75 Blank Line**

1554 A line consisting solely of zero or more <blank>s terminated by a <newline>; see also Section  
1555 3.144 (on page 55).

### 1556 **3.76 Blocked Process (or Thread)**

1557 A process (or thread) that is waiting for some condition (other than the availability of a  
1558 processor) to be satisfied before it can continue execution.

### 1559 **3.77 Blocking**

1560 A property of an open file description that causes function calls associated with it to wait for the  
1561 requested action to be performed before returning.

### 1562 **3.78 Block-Mode Terminal**

1563 A terminal device operating in a mode incapable of the character-at-a-time input and output  
1564 operations described by some of the standard utilities.

1565 **Note:** Output Devices and Terminal Types are defined in detail in Section 10.2 (on page 185).

### 1566 3.79 Block Special File

1567 A file that refers to a device. A block special file is normally distinguished from a character  
1568 special file by providing access to the device in a manner such that the hardware characteristics  
1569 of the device are not visible.

### 1570 3.80 Braces

1571 The characters ' { ' (left brace) and ' } ' (right brace), also known as curly braces. When used in  
1572 the phrase “enclosed in (curly) braces” the symbol ' { ' immediately precedes the object to be  
1573 enclosed, and ' } ' immediately follows it. When describing these characters in the portable  
1574 character set, the names <left-brace> and <right-brace> are used.

### 1575 3.81 Brackets

1576 The characters ' [ ' (left-bracket) and ' ] ' (right-bracket), also known as square brackets. When  
1577 used in the phrase “enclosed in (square) brackets” the symbol ' [ ' immediately precedes the  
1578 object to be enclosed, and ' ] ' immediately follows it. When describing these characters in the  
1579 portable character set, the names <left-square-bracket> and <right-square-bracket> are used.

### 1580 3.82 Broadcast

1581 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and  
1582 RFC 922.

### 1583 3.83 Built-In Utility (or Built-In)

1584 A utility implemented within a shell. The utilities referred to as special built-ins have special  
1585 qualities. Unless qualified, the term “built-in” includes the special built-in utilities. Regular  
1586 built-ins are not required to be actually built into the shell on the implementation, but they do  
1587 have special command-search qualities.

1588 **Note:** Special Built-In Utilities are defined in detail in the Shell and Utilities volume of  
1589 IEEE Std 1003.1-2001, Section 2.14, Special Built-In Utilities.

1590 Regular Built-In Utilities are defined in detail in the Shell and Utilities volume of  
1591 IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

### 1592 3.84 Byte

1593 An individually addressable unit of data storage that is exactly an octet, used to store a character  
1594 or a portion of a character; see also Section 3.87 (on page 47). A byte is composed of a  
1595 contiguous sequence of 8 bits. The least significant bit is called the “low-order” bit; the most  
1596 significant is called the “high-order” bit.

1597 **Note:** The definition of byte from the ISO C standard is broader than the above and might  
1598 accommodate hardware architectures with different sized addressable units than octets.

### 1599 3.85 Byte Input/Output Functions

1600 The functions that perform byte-oriented input from streams or byte-oriented output to streams:  
 1601 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *printf()*,  
 1602 *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1603 **Note:** Functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-2001.

### 1604 3.86 Carriage-Return Character (<carriage-return>)

1605 A character that in the output stream indicates that printing should start at the beginning of the  
 1606 same physical line in which the <carriage-return> occurred. It is the character designated by  
 1607 '␣' in the C language. It is unspecified whether this character is the exact sequence  
 1608 transmitted to an output device by the system to accomplish the movement to the beginning of  
 1609 the line.

### 1610 3.87 Character

1611 A sequence of one or more bytes representing a single graphic symbol or control code.

1612 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte  
 1613 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,  
 1614 *character* here has no necessary relationship with storage space, and *byte* is used when storage  
 1615 space is discussed.

1616 See the definition of the portable character set in Section 6.1 (on page 113) for a further  
 1617 explanation of the graphical representations of (abstract) characters, as opposed to character  
 1618 encodings.

### 1619 3.88 Character Array

1620 An array of elements of type **char**.

### 1621 3.89 Character Class

1622 A named set of characters sharing an attribute associated with the name of the class. The classes  
 1623 and the characters that they contain are dependent on the value of the *LC\_CTYPE* category in the  
 1624 current locale.

1625 **Note:** The *LC\_CTYPE* category is defined in detail in Section 7.3.1 (on page 126).

### 1626 3.90 Character Set

1627 A finite set of different characters used for the representation, organization, or control of data.

### 1628 **3.91 Character Special File**

1629 A file that refers to a device. One specific type of character special file is a terminal device file.

1630 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 187).

### 1631 **3.92 Character String**

1632 A contiguous sequence of characters terminated by and including the first null byte.

### 1633 **3.93 Child Process**

1634 A new process created (by *fork()*, *posix\_spawn()*, *posix\_spawnp()*, or *vfork()*) by a given process. 2

1635 A child process remains the child of the creating process as long as both processes continue to exist.

1636  
1637 **Note:** The *fork()*, *posix\_spawn()*, *posix\_spawnp()*, and *vfork()* functions are defined in detail in the 2  
1638 System Interfaces volume of IEEE Std 1003.1-2001.

### 1639 **3.94 Circumflex**

1640 The character '^'.

### 1641 **3.95 Clock**

1642 A software or hardware object that can be used to measure the apparent or actual passage of time.

1643  
1644 The current value of the time measured by a clock can be queried and, possibly, set to a value within the legal range of the clock.

### 1646 **3.96 Clock Jump**

1647 The difference between two successive distinct values of a clock, as observed from the application via one of the “get time” operations.

### 1649 **3.97 Clock Tick**

1650 An interval of time; an implementation-defined number of these occur each second. Clock ticks are one of the units that may be used to express a value found in type **clock\_t**.

### 1652 **3.98 Coded Character Set**

1653 A set of unambiguous rules that establishes a character set and the one-to-one relationship between each character of the set and its bit representation.

### 1655 3.99 Codeset

1656 The result of applying rules that map a numeric code value to each element of a character set. An  
 1657 element of a character set may be related to more than one numeric code value but the reverse is  
 1658 not true. However, for state-dependent encodings the relationship between numeric code values  
 1659 and elements of a character set may be further controlled by state information. The character set  
 1660 may contain fewer elements than the total number of possible numeric code values; that is, some  
 1661 code values may be unassigned.

1662 **Note:** Character Encoding is defined in detail in Section 6.2 (on page 116).

### 1663 3.100 Collating Element

1664 The smallest entity used to determine the logical ordering of character or wide-character strings;  
 1665 see also Section 3.102. A collating element consists of either a single character, or two or more  
 1666 characters collating as a single entity. The value of the *LC\_COLLATE* category in the current  
 1667 locale determines the current set of collating elements.

### 1668 3.101 Collation

1669 The logical ordering of character or wide-character strings according to defined precedence  
 1670 rules. These rules identify a collation sequence between the collating elements, and such  
 1671 additional rules that can be used to order strings consisting of multiple collating elements.

### 1672 3.102 Collation Sequence

1673 The relative order of collating elements as determined by the setting of the *LC\_COLLATE*  
 1674 category in the current locale. The collation sequence is used for sorting and is determined from  
 1675 the collating weights assigned to each collating element. In the absence of weights, the collation  
 1676 sequence is the order in which collating elements are specified between **order\_start** and  
 1677 **order\_end** keywords in the *LC\_COLLATE* category.

1678 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to  
 1679 the limit {*COLL\_WEIGHTS\_MAX*}. On each level, elements may be given the same weight (at  
 1680 the primary level, called an equivalence class; see also Section 3.150 (on page 55)) or be omitted  
 1681 from the sequence. Strings that collate equally using the first assigned weight (primary ordering)  
 1682 are then compared using the next assigned weight (secondary ordering), and so on.

1683 **Note:** {*COLL\_WEIGHTS\_MAX*} is defined in detail in <**limits.h**>.

### 1684 3.103 Column Position

1685 A unit of horizontal measure related to characters in a line.

1686 It is assumed that each character in a character set has an intrinsic column width independent of  
 1687 any output device. Each printable character in the portable character set has a column width of  
 1688 one. The standard utilities, when used as described in IEEE Std 1003.1-2001, assume that all  
 1689 characters have integral column widths. The column width of a character is not necessarily  
 1690 related to the internal representation of the character (numbers of bits or bytes).

1691 The column position of a character in a line is defined as one plus the sum of the column widths  
 1692 of the preceding characters in the line. Column positions are numbered starting from 1.

**1693 3.104 Command**

1694 A directive to the shell to perform a particular task.

1695 **Note:** Shell Commands are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001,  
1696 Section 2.9, Shell Commands.

**1697 3.105 Command Language Interpreter**

1698 An interface that interprets sequences of text input as commands. It may operate on an input  
1699 stream or it may interactively prompt and read commands from a terminal. It is possible for  
1700 applications to invoke utilities through a number of interfaces, which are collectively considered  
1701 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*  
1702 function, although *popen()* and the various forms of *exec* may also be considered to behave as  
1703 interpreters.

1704 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

1705 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume  
1706 of IEEE Std 1003.1-2001.

**1707 3.106 Composite Graphic Symbol**

1708 A graphic symbol consisting of a combination of two or more other graphic symbols in a single  
1709 character position, such as a diacritical mark and a base character.

**1710 3.107 Condition Variable**

1711 A synchronization object which allows a thread to suspend execution, repeatedly, until some  
1712 associated predicate becomes true. A thread whose execution is suspended on a condition  
1713 variable is said to be blocked on the condition variable.

**1714 3.108 Connection**

1715 An association established between two or more endpoints for the transfer of data

**1716 3.109 Connection Mode**

1717 The transfer of data in the context of a connection; see also Section 3.110.

**1718 3.110 Connectionless Mode**

1719 The transfer of data other than in the context of a connection; see also Section 3.109 and Section  
1720 3.123 (on page 52).



**1721 3.111 Control Character**

1722 A character, other than a graphic character, that affects the recording, processing, transmission,  
1723 or interpretation of text.

**1724 3.112 Control Operator**

1725 In the shell command language, a token that performs a control function. It is one of the  
1726 following symbols:

1727        &    &&    (    )    ;    ;;    newline    |    ||

1728 The end-of-input indicator used internally by the shell is also considered a control operator.

1729 **Note:**      Token Recognition is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001,  
1730              Section 2.3, Token Recognition.

**1731 3.113 Controlling Process**

1732 The session leader that established the connection to the controlling terminal. If the terminal  
1733 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be  
1734 the controlling process.

**1735 3.114 Controlling Terminal**

1736 A terminal that is associated with a session. Each session may have at most one controlling  
1737 terminal associated with it, and a controlling terminal is associated with exactly one session.  
1738 Certain input sequences from the controlling terminal cause signals to be sent to all processes in  
1739 the process group associated with the controlling terminal.

1740 **Note:**      The General Terminal Interface is defined in detail in Chapter 11 (on page 187).

**1741 3.115 Conversion Descriptor**

1742 A per-process unique value used to identify an open codeset conversion.

**1743 3.116 Core File**

1744 A file of unspecified format that may be generated when a process terminates abnormally.

**1745 3.117 CPU Time (Execution Time)**

1746 The time spent executing a process or thread, including the time spent executing system services  
1747 on behalf of that process or thread. If the Threads option is supported, then the value of the  
1748 CPU-time clock for a process is implementation-defined. With this definition the sum of all the  
1749 execution times of all the threads in a process might not equal the process execution time, even  
1750 in a single-threaded process, because implementations may differ in how they account for time  
1751 during context switches or for other reasons.

**1752 3.118 CPU-Time Clock**

1753 A clock that measures the execution time of a particular process or thread.

**1754 3.119 CPU-Time Timer**

1755 A timer attached to a CPU-time clock.

**1756 3.120 Current Job**

1757 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There  
1758 is at most one current job; see also Section 3.203 (on page 63).

**1759 3.121 Current Working Directory**

1760 See *Working Directory* in Section 3.436 (on page 95).

**1761 3.122 Cursor Position**

1762 The line and column position on the screen denoted by the terminal's cursor.

**1763 3.123 Datagram**

1764 A unit of data transferred from one endpoint to another in connectionless mode service.

**1765 3.124 Data Segment**

1766 Memory associated with a process, that can contain dynamically allocated data.

**1767 3.125 Deferred Batch Service**

1768 A service that is performed as a result of events that are asynchronous with respect to requests.

1769 **Note:** Once a batch job has been created, it is subject to deferred services.

**1770 3.126 Device**

1771 A computer peripheral or an object that appears to the application as such.

**1772 3.127 Device ID**

1773 A non-negative integer used to identify a device.

**1774 3.128 Directory**

1775 A file that contains directory entries. No two directory entries in the same directory have the  
1776 same name.

**1777 3.129 Directory Entry (or Link)**

1778 An object that associates a filename with a file. Several directory entries can associate names  
1779 with the same file.

**1780 3.130 Directory Stream**

1781 A sequence of all the directory entries in a particular directory. An open directory stream may be  
1782 implemented using a file descriptor.

**1783 3.131 Disarm (a Timer)**

1784 To stop a timer from measuring the passage of time, disabling any future process notifications  
1785 (until the timer is armed again).

**1786 3.132 Display**

1787 To output to the user's terminal. If the output is not directed to a terminal, the results are  
1788 undefined.

**1789 3.133 Display Line**

1790 A line of text on a physical device or an emulation thereof. Such a line will have a maximum  
1791 number of characters which can be presented.

1792 **Note:** This may also be written as “line on the display”.

**1793 3.134 Dollar Sign**

1794 The character ' \$ '.

**1795 3.135 Dot**

1796 In the context of naming files, the filename consisting of a single dot character ( ' . ' ).

1797 **Note:** In the context of shell special built-in utilities, see *dot* in the Shell and Utilities volume of  
1798 IEEE Std 1003.1-2001, Section 2.14, Special Built-In Utilities.

1799 Pathname Resolution is defined in detail in Section 4.11 (on page 100).

**1800 3.136 Dot-Dot**

1801 The filename consisting solely of two dot characters (" . . ").

1802 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 100).

**1803 3.137 Double-Quote**

1804 The character ' " ', also known as quotation-mark.

1805 **Note:** The “double” adjective in this term refers to the two strokes in the character glyph.  
1806 IEEE Std 1003.1-2001 never uses the term “double-quote” to refer to two apostrophes or  
1807 quotation marks.

**1808 3.138 Downshifting**

1809 The conversion of an uppercase character that has a single-character lowercase representation  
1810 into this lowercase representation.

**1811 3.139 Driver**

1812 A module that controls data transferred to and received from devices.

1813 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are  
1814 frequently written separately from the writing of the implementation. A driver may contain  
1815 processor-specific code, and therefore be non-portable.

**1816 3.140 Effective Group ID**

1817 An attribute of a process that is used in determining various permissions, including file access  
1818 permissions; see also Section 3.188 (on page 61).

**1819 3.141 Effective User ID**

1820 An attribute of a process that is used in determining various permissions, including file access  
1821 permissions; see also Section 3.425 (on page 93).

**1822 3.142 Eight-Bit Transparency**

1823 The ability of a software component to process 8-bit characters without modifying or utilizing  
1824 any part of the character in a way that is inconsistent with the rules of the current coded  
1825 character set.

**1826 3.143 Empty Directory**

1827 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link  
1828 to it, in dot-dot. No other links to the directory may exist. It is unspecified whether an  
1829 implementation can ever consider the root directory to be empty.

**1830 3.144 Empty Line**

1831 A line consisting of only a <newline>; see also Section 3.75 (on page 45).

**1832 3.145 Empty String (or Null String)**

1833 A string whose first byte is a null byte.

**1834 3.146 Empty Wide-Character String**

1835 A wide-character string whose first element is a null wide-character code.

**1836 3.147 Encoding Rule**

1837 The rules used to convert between wide-character codes and multi-byte character codes.

1838 **Note:** Stream Orientation and Encoding Rules are defined in detail in the System Interfaces volume  
1839 of IEEE Std 1003.1-2001, Section 2.5.2, Stream Orientation and Encoding Rules.

**1840 3.148 Entire Regular Expression**

1841 The concatenated set of one or more basic regular expressions or extended regular expressions  
1842 that make up the pattern specified for string selection.

1843 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 169).

**1844 3.149 Epoch**

1845 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time  
1846 (UTC).

1847 **Note:** See also Seconds Since the Epoch defined in Section 4.14 (on page 102).

**1848 3.150 Equivalence Class**

1849 A set of collating elements with the same primary collation weight.

1850 Elements in an equivalence class are typically elements that naturally group together, such as all  
1851 accented letters based on the same base letter.

1852 The collation order of elements within an equivalence class is determined by the weights  
1853 assigned on any subsequent levels after the primary weight.

**1854 3.151 Era**

1855 A locale-specific method for counting and displaying years.

1856 **Note:** The *LC\_TIME* category is defined in detail in Section 7.3.5 (on page 147).

**1857 3.152 Event Management**

1858 The mechanism that enables applications to register for and be made aware of external events  
1859 such as data becoming available for reading.

**1860 3.153 Executable File**

1861 A regular file acceptable as a new process image file by the equivalent of the *exec* family of  
1862 functions, and thus usable as one form of a utility. The standard utilities described as compilers  
1863 can produce executable files, but other unspecified methods of producing executable files may  
1864 also be provided. The internal format of an executable file is unspecified, but a conforming  
1865 application cannot assume an executable file is a text file.

**1866 3.154 Execute**

1867 To perform command search and execution actions, as defined in the Shell and Utilities volume  
1868 of IEEE Std 1003.1-2001; see also Section 3.200 (on page 62).

1869 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume of  
1870 IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

**1871 3.155 Execution Time**

1872 See *CPU Time* in Section 3.117 (on page 51).

**1873 3.156 Execution Time Monitoring**

1874 A set of execution time monitoring primitives that allow online measuring of thread and process  
1875 execution times.

**1876 3.157 Expand**

1877 In the shell command language, when not qualified, the act of applying word expansions.

1878 **Note:** Word Expansions are defined in detail in the Shell and Utilities volume of  
1879 IEEE Std 1003.1-2001, Section 2.6, Word Expansions.

**1880 3.158 Extended Regular Expression (ERE)**

1881 A regular expression (see also Section 3.316 (on page 79)) that is an alternative to the Basic  
1882 Regular Expression using a more extensive syntax, occasionally used by some utilities.

1883 **Note:** Extended Regular Expressions are described in detail in Section 9.4 (on page 175).

**1884 3.159 Extended Security Controls**

1885 Implementation-defined security controls allowed by the file access permission and appropriate  
1886 privilege (see also Section 3.19 (on page 37)) mechanisms, through which an implementation can  
1887 support different security policies from those described in IEEE Std 1003.1-2001.

1888 **Note:** See also Extended Security Controls defined in Section 4.3 (on page 97).

1889 File Access Permissions are defined in detail in Section 4.4 (on page 97).

**1890 3.160 Feature Test Macro**

1891 A macro used to determine whether a particular set of features is included from a header.

1892 **Note:** See also the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.2, The Compilation  
1893 Environment.

**1894 3.161 Field**

1895 In the shell command language, a unit of text that is the result of parameter expansion,  
1896 arithmetic expansion, command substitution, or field splitting. During command processing, the  
1897 resulting fields are used as the command name and its arguments.

1898 **Note:** Parameter Expansion is defined in detail in the Shell and Utilities volume of  
1899 IEEE Std 1003.1-2001, Section 2.6.2, Parameter Expansion.

1900 Arithmetic Expansion is defined in detail in the Shell and Utilities volume of  
1901 IEEE Std 1003.1-2001, Section 2.6.4, Arithmetic Expansion.

1902 Command Substitution is defined in detail in the Shell and Utilities volume of  
1903 IEEE Std 1003.1-2001, Section 2.6.3, Command Substitution.

1904 Field Splitting is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001,  
1905 Section 2.6.5, Field Splitting.

1906 For further information on command processing, see the Shell and Utilities volume of  
1907 IEEE Std 1003.1-2001, Section 2.9.1, Simple Commands.

**1908 3.162 FIFO Special File (or FIFO)**

1909 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

1910 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of  
1911 IEEE Std 1003.1-2001, *lseek()*, *open()*, *read()*, and *write()*.

**1912 3.163 File**

1913 An object that can be written to, or read from, or both. A file has certain attributes, including  
1914 access permissions and type. File types include regular file, character special file, block special  
1915 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported  
1916 by the implementation.

1917 **3.164 File Description**1918 See *Open File Description* in Section 3.253 (on page 70).1919 **3.165 File Descriptor**

1920 A per-process unique, non-negative integer used to identify an open file for the purpose of file  
 1921 access. The value of a file descriptor is from zero to {OPEN\_MAX}. A process can have no more  
 1922 than {OPEN\_MAX} file descriptors open simultaneously. File descriptors may also be used to  
 1923 implement message catalog descriptors and directory streams; see also Section 3.253 (on page  
 1924 70).

1925 **Note:** {OPEN\_MAX} is defined in detail in <limits.h>.1926 **3.166 File Group Class**

1927 The property of a file indicating access permissions for a process related to the group  
 1928 identification of a process. A process is in the file group class of a file if the process is not in the  
 1929 file owner class and if the effective group ID or one of the supplementary group IDs of the  
 1930 process matches the group ID associated with the file. Other members of the class may be  
 1931 implementation-defined.

1932 **3.167 File Mode**

1933 An object containing the file mode bits and file type of a file.

1934 **Note:** File mode bits and file types are defined in detail in <sys/stat.h>.1935 **3.168 File Mode Bits**

1936 A file's file permission bits: set-user-ID-on-execution bit (S\_ISUID), set-group-ID-on-execution  
 1937 bit (S\_ISGID), and, on directories, the restricted deletion flag bit (S\_ISVTX).

1938 **Note:** File Mode Bits are defined in detail in <sys/stat.h>.1939 **3.169 Filename**

1940 A name consisting of 1 to {NAME\_MAX} bytes used to name a file. The characters composing  
 1941 the name may be selected from the set of all character values excluding the slash character and  
 1942 the null byte. The filenames dot and dot-dot have special meaning. A filename is sometimes  
 1943 referred to as a "pathname component".

1944 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 100).1945 **3.170 Filename Portability**

1946 Filenames should be constructed from the portable filename character set because the use of  
 1947 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a  
 1948 colon (':') in a pathname could cause ambiguity if that pathname were included in a *PATH*  
 1949 definition.)



**1950 3.171 File Offset**

1951 The byte position in the file where the next I/O operation begins. Each open file description  
1952 associated with a regular file, block special file, or directory has a file offset. A character special  
1953 file that does not refer to a terminal device may have a file offset. There is no file offset specified  
1954 for a pipe or FIFO.

**1955 3.172 File Other Class**

1956 The property of a file indicating access permissions for a process related to the user and group  
1957 identification of a process. A process is in the file other class of a file if the process is not in the  
1958 file owner class or file group class.

**1959 3.173 File Owner Class**

1960 The property of a file indicating access permissions for a process related to the user  
1961 identification of a process. A process is in the file owner class of a file if the effective user ID of  
1962 the process matches the user ID of the file.

**1963 3.174 File Permission Bits**

1964 Information about a file that is used, along with other information, to determine whether a  
1965 process has read, write, or execute/search permission to a file. The bits are divided into three  
1966 parts: owner, group, and other. Each part is used with the corresponding file class of processes.  
1967 These bits are contained in the file mode.

1968 **Note:** File modes are defined in detail in <sys/stat.h>.

1969 File Access Permissions are defined in detail in Section 4.4 (on page 97).

**1970 3.175 File Serial Number**

1971 A per-file system unique identifier for a file.

**1972 3.176 File System**

1973 A collection of files and certain of their attributes. It provides a name space for file serial  
1974 numbers referring to those files.

**1975 3.177 File Type**

1976 See *File* in Section 3.163 (on page 57).

1977 **3.178 Filter**

1978 A command whose operation consists of reading data from standard input or a list of input files  
 1979 and writing data to standard output. Typically, its function is to perform some transformation  
 1980 on the data stream.

1981 **3.179 First Open (of a File)**

1982 When a process opens a file that is not currently an open file within any process.

1983 **3.180 Flow Control**

1984 The mechanism employed by a communications provider that constrains a sending entity to  
 1985 wait until the receiving entities can safely receive additional data without loss.

1986 **3.181 Foreground Job**

1987 See *Foreground Process Group* in Section 3.183.

1988 **3.182 Foreground Process**

1989 A process that is a member of a foreground process group.

1990 **3.183 Foreground Process Group (or Foreground Job)**

1991 A process group whose member processes have certain privileges, denied to processes in  
 1992 background process groups, when accessing their controlling terminal. Each session that has  
 1993 established a connection with a controlling terminal has at most one process group of the session  
 1994 as the foreground process group of that controlling terminal.

1995 **Note:** The General Terminal Interface is defined in detail in Chapter 11.

1996 **3.184 Foreground Process Group ID**

1997 The process group ID of the foreground process group.

1998 **3.185 Form-Feed Character (<form-feed>)**

1999 A character that in the output stream indicates that printing should start on the next page of an  
 2000 output device. It is the character designated by '`\f`' in the C language. If the <form-feed> is not  
 2001 the first character of an output line, the result is unspecified. It is unspecified whether this  
 2002 character is the exact sequence transmitted to an output device by the system to accomplish the  
 2003 movement to the next page.

2004 **3.186 Graphic Character**2005 A member of the **graph** character class of the current locale.2006 **Note:** The **graph** character class is defined in detail in Section 7.3.1 (on page 126).2007 **3.187 Group Database**

2008 A system database that contains at least the following information for each group ID: 2

- 2009 • Group name
- 2010 • Numerical group ID
- 2011 • List of users allowed in the group

2012 The list of users allowed in the group is used by the *newgrp* utility.2013 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.2014 **3.188 Group ID**

2015 A non-negative integer, which can be contained in an object of type **gid\_t**, that is used to identify

2016 a group of system users. Each system user is a member of at least one group. When the identity

2017 of a group is associated with a process, a group ID value is referred to as a real group ID, an

2018 effective group ID, one of the supplementary group IDs, or a saved set-group-ID.

2019 **3.189 Group Name**

2020 A string that is used to identify a group; see also Section 3.187. To be portable across conforming

2021 systems, the value is composed of characters from the portable filename character set. The

2022 hyphen should not be used as the first character of a portable group name.

2023 **3.190 Hard Limit**

2024 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.

2025 A non-privileged process is restricted to only lowering its hard limit.

2026 **3.191 Hard Link**

2027 The relationship between two directory entries that represent the same file; see also Section 3.129

2028 (on page 53). The result of an execution of the *ln* utility (without the *-s* option) or the *link()*

2029 function. This term is contrasted against symbolic link; see also Section 3.372 (on page 86).

2030 **3.192 Home Directory**2031 The directory specified by the *HOME* environment variable.

**2032 3.193 Host Byte Order**

2033 The arrangement of bytes in any integer type when using a specific machine architecture.

2034 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian is a  
2035 format for storage of binary data in which the most significant byte is placed first, with the rest  
2036 in descending order. Little-endian is a format for storage or transmission of binary data in  
2037 which the least significant byte is placed first, with the rest in ascending order. See also Section  
2038 4.8 (on page 99).

**2039 3.194 Incomplete Line**

2040 A sequence of one or more non-<newline>s at the end of the file.

**2041 3.195 Inf**

2042 A value representing +infinity or a value representing –infinity that can be stored in a floating  
2043 type. Not all systems support the Inf values.

**2044 3.196 Instrumented Application**

2045 An application that contains at least one call to the trace point function *posix\_trace\_event()*. Each  
2046 process of an instrumented application has a mapping of trace event names to trace event type  
2047 identifiers. This mapping is used by the trace stream that is created for that process.

**2048 3.197 Interactive Shell**

2049 A processing mode of the shell that is suitable for direct user interaction.

**2050 3.198 Internationalization**

2051 The provision within a computer program of the capability of making itself adaptable to the  
2052 requirements of different native languages, local customs, and coded character sets.

**2053 3.199 Interprocess Communication**

2054 A functionality enhancement to add a high-performance, deterministic interprocess  
2055 communication facility for local communication.

**2056 3.200 Invoke**

2057 To perform command search and execution actions, except that searching for shell functions and  
2058 special built-in utilities is suppressed; see also Section 3.154 (on page 56).

2059 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume of  
2060 IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

2061 **3.201 Job**

2062 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all  
 2063 in the same process group.

2064 **Note:** See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.9.2, Pipelines.

2065 **3.202 Job Control**

2066 A facility that allows users selectively to stop (suspend) the execution of processes and continue  
 2067 (resume) their execution at a later point. The user typically employs this facility via the  
 2068 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

2069 **3.203 Job Control Job ID**

2070 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in the  
 2071 following table:

2072 **Table 3-1 Job Control Job ID Formats**

2073 2074 Job Control Job ID	Meaning
2075 %%	Current job.
2076 %+	Current job.
2077 %-	Previous job.
2078 % <i>n</i>	Job number <i>n</i> .
2079 % <i>string</i>	Job whose command begins with <i>string</i> .
2080 %? <i>string</i>	Job whose command contains <i>string</i> .

2081 **3.204 Last Close (of a File)**

2082 When a process closes a file, resulting in the file not being an open file within any process.

2083 **3.205 Line**

2084 A sequence of zero or more non-`<newline>`s plus a terminating `<newline>`.

2085 **3.206 Linger**

2086 A period of time before terminating a connection, to allow outstanding data to be transferred.

2087 **3.207 Link**

2088 See *Directory Entry* in Section 3.129 (on page 53).

**2089 3.208 Link Count**

2090 The number of directory entries that refer to a particular file.

**2091 3.209 Local Customs**

2092 The conventions of a geographical area or territory for such things as date, time, and currency  
2093 formats.

**2094 3.210 Local Interprocess Communication (Local IPC)**

2095 The transfer of data between processes in the same system.

**2096 3.211 Locale**

2097 The definition of the subset of a user's environment that depends on language and cultural  
2098 conventions.

2099 **Note:** Locales are defined in detail in Chapter 7 (on page 123).

**2100 3.212 Localization**

2101 The process of establishing information within a computer system specific to the operation of  
2102 particular native languages, local customs, and coded character sets.

**2103 3.213 Login**

2104 The unspecified activity by which a user gains access to the system. Each login is associated  
2105 with exactly one login name.

**2106 3.214 Login Name**

2107 A user name that is associated with a login.

**2108 3.215 Map**

2109 To create an association between a page-aligned range of the address space of a process and  
2110 some memory object, such that a reference to an address in that range of the address space  
2111 results in a reference to the associated memory object. The mapped memory object is not  
2112 necessarily memory-resident.

### 2113 3.216 Marked Message

2114 A STREAMS message on which a certain flag is set. Marking a message gives the application  
2115 protocol-specific information. An application can use *ioctl()* to determine whether a given  
2116 message is marked.

2117 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of  
2118 IEEE Std 1003.1-2001.

### 2119 3.217 Matched

2120 A state applying to a sequence of zero or more characters when the characters in the sequence  
2121 correspond to a sequence of characters defined by a basic regular expression or extended regular  
2122 expression pattern.

2123 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 169).

### 2124 3.218 Memory Mapped Files

2125 A facility to allow applications to access files as part of the address space.

### 2126 3.219 Memory Object

2127 One of:

- 2128 • A file (see Section 3.163 (on page 57))
- 2129 • A shared memory object (see Section 3.340 (on page 82))
- 2130 • A typed memory object (see Section 3.418 (on page 92))

2131 When used in conjunction with *mmap()*, a memory object appears in the address space of the  
2132 calling process.

2133 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of  
2134 IEEE Std 1003.1-2001.

### 2135 3.220 Memory-Resident

2136 The process of managing the implementation in such a way as to provide an upper bound on  
2137 memory access times.

### 2138 3.221 Message

2139 In the context of programmatic message passing, information that can be transferred between  
2140 processes or threads by being added to and removed from a message queue. A message consists  
2141 of a fixed-size message buffer.

**2142 3.222 Message Catalog**

2143 In the context of providing natural language messages to the user, a file or storage area  
2144 containing program messages, command prompts, and responses to prompts for a particular  
2145 native language, territory, and codeset.

**2146 3.223 Message Catalog Descriptor**

2147 In the context of providing natural language messages to the user, a per-process unique value  
2148 used to identify an open message catalog. A message catalog descriptor may be implemented  
2149 using a file descriptor.

**2150 3.224 Message Queue**

2151 In the context of programmatic message passing, an object to which messages can be added and  
2152 removed. Messages may be removed in the order in which they were added or in priority order.

**2153 3.225 Mode**

2154 A collection of attributes that specifies a file's type and its access permissions.

2155 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 97).

**2156 3.226 Monotonic Clock**

2157 A clock whose value cannot be set via `clock_settime()` and which cannot have negative clock  
2158 jumps.

**2159 3.227 Mount Point**

2160 Either the system root directory or a directory for which the `st_dev` field of structure `stat` differs  
2161 from that of its parent directory.

2162 **Note:** The `stat` structure is defined in detail in `<sys/stat.h>`.

**2163 3.228 Multi-Character Collating Element**

2164 A sequence of two or more characters that collate as an entity. For example, in some coded  
2165 character sets, an accented character is represented by a non-spacing accent, followed by the  
2166 letter. Other examples are the Spanish elements *ch* and *ll*.

**2167 3.229 Mutex**

2168 A synchronization object used to allow multiple threads to serialize their access to shared data.  
2169 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has  
2170 locked a mutex becomes its owner and remains the owner until that same thread unlocks the  
2171 mutex.



**2172 3.230 Name**

2173 In the shell command language, a word consisting solely of underscores, digits, and alphabets  
2174 from the portable character set. The first character of a name is not a digit.

2175 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 113).

**2176 3.231 Named STREAM**

2177 A STREAMS-based file descriptor that is attached to a name in the file system name space. All  
2178 subsequent operations on the named STREAM act on the STREAM that was associated with the  
2179 file descriptor until the name is disassociated from the STREAM.

**2180 3.232 NaN (Not a Number)**

2181 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-  
2182 point numbers. Not all systems support NaN values.

**2183 3.233 Native Language**

2184 A computer user's spoken or written language, such as American English, British English,  
2185 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

**2186 3.234 Negative Response**

2187 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category  
2188 keyword **noexpr**, matching an extended regular expression in the current locale.

2189 **Note:** The *LC\_MESSAGES* category is defined in detail in Section 7.3.6 (on page 152).

**2190 3.235 Network**

2191 A collection of interconnected hosts.

2192 **Note:** The term “network” in IEEE Std 1003.1-2001 is used to refer to the network of hosts. The term  
2193 “batch system” is used to refer to the network of batch servers.

**2194 3.236 Network Address**

2195 A network-visible identifier used to designate specific endpoints in a network. Specific  
2196 endpoints on host systems have addresses, and host systems may also have addresses.

### 2197 **3.237 Network Byte Order**

2198 The way of representing any integer type such that, when transmitted over a network via a  
 2199 network endpoint, the **int** type is transmitted as an appropriate number of octets with the most  
 2200 significant octet first, followed by any other octets in descending order of significance.

2201 **Note:** This order is more commonly known as big-endian ordering. See also Section 4.8 (on page 99).

### 2202 **3.238 Newline Character (<newline>)**

2203 A character that in the output stream indicates that printing should start at the beginning of the  
 2204 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this  
 2205 character is the exact sequence transmitted to an output device by the system to accomplish the  
 2206 movement to the next line.

### 2207 **3.239 Nice Value**

2208 A number used as advice to the system to alter process scheduling. Numerically smaller values  
 2209 give a process additional preference when scheduling a process to run. Numerically larger  
 2210 values reduce the preference and make a process less likely to run. Typically, a process with a  
 2211 smaller nice value runs to completion more quickly than an equivalent process with a higher  
 2212 nice value. The symbol {NZERO} specifies the default nice value of the system.

### 2213 **3.240 Non-Blocking**

2214 A property of an open file description that causes function calls involving it to return without  
 2215 delay when it is detected that the requested action associated with the function call cannot be  
 2216 completed without unknown delay.

2217 **Note:** The exact semantics are dependent on the type of file associated with the open file description.  
 2218 For data reads from devices such as ttys and FIFOs, this property causes the read to return  
 2219 immediately when no data was available. Similarly, for writes, it causes the call to return  
 2220 immediately when the thread would otherwise be delayed in the write operation; for example,  
 2221 because no space was available. For networking, it causes functions not to await protocol  
 2222 events (for example, acknowledgements) to occur. See also the System Interfaces volume of  
 2223 IEEE Std 1003.1-2001, Section 2.10.7, Socket I/O Mode.

### 2224 **3.241 Non-Spacing Characters**

2225 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:2001 2  
 2226 standard coded character set, which is used in combination with other characters to form  
 2227 composite graphic symbols.

### 2228 **3.242 NUL**

2229 A character with all bits set to zero.

**2230 3.243 Null Byte**

2231 A byte with all bits set to zero.

**2232 3.244 Null Pointer**

2233 The value that is obtained by converting the number 0 into a pointer; for example, **(void \*) 0**. The  
2234 C language guarantees that this value does not match that of any legitimate pointer, so it is used  
2235 by many functions that return pointers to indicate an error.

**2236 3.245 Null String**

2237 See *Empty String* in Section 3.145 (on page 55).

**2238 3.246 Null Wide-Character Code**

2239 A wide-character code with all bits set to zero.

**2240 3.247 Number Sign**

2241 The character ' # ', also known as hash sign.

**2242 3.248 Object File**

2243 A regular file containing the output of a compiler, formatted as input to a linkage editor for  
2244 linking with other object files into an executable form. The methods of linking are unspecified  
2245 and may involve the dynamic linking of objects at runtime. The internal format of an object file  
2246 is unspecified, but a conforming application cannot assume an object file is a text file.

**2247 3.249 Octet**

2248 Unit of data representation that consists of eight contiguous bits.

**2249 3.250 Offset Maximum**

2250 An attribute of an open file description representing the largest value that can be used as a file  
2251 offset.

**2252 3.251 Opaque Address**

2253 An address such that the entity making use of it requires no details about its contents or format.

**2254 3.252 Open File**

2255 A file that is currently associated with a file descriptor.

**2256 3.253 Open File Description**

2257 A record of how a process or group of processes is accessing a file. Each file descriptor refers to  
2258 exactly one open file description, but an open file description can be referred to by more than  
2259 one file descriptor. The file offset, file status, and file access modes are attributes of an open file  
2260 description.

**2261 3.254 Operand**

2262 An argument to a command that is generally used as an object supplying information to a utility  
2263 necessary to complete its processing. Operands generally follow the options in a command line.

2264 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 201).

**2265 3.255 Operator**

2266 In the shell command language, either a control operator or a redirection operator.

**2267 3.256 Option**

2268 An argument to a command that is generally used to specify changes in the utility's default  
2269 behavior.

2270 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 201).

**2271 3.257 Option-Argument**

2272 A parameter that follows certain options. In some cases an option-argument is included within  
2273 the same argument string as the option—in most cases it is the next argument.

2274 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 201).

**2275 3.258 Orientation**

2276 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2277 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-2001, Section  
2278 2.5.2, Stream Orientation and Encoding Rules.

**2279 3.259 Orphaned Process Group**

2280 A process group in which the parent of every member is either itself a member of the group or is  
2281 not a member of the group's session.

**2282 3.260 Page**

2283 The granularity of process memory mapping or locking.

2284 Physical memory and memory objects can be mapped into the address space of a process on  
2285 page boundaries and in integral multiples of pages. Process address space can be locked into  
2286 memory (made memory-resident) on page boundaries and in integral multiples of pages.

**2287 3.261 Page Size**

2288 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On systems  
2289 that have segment rather than page-based memory architectures, the term “page” means a  
2290 segment.

**2291 3.262 Parameter**

2292 In the shell command language, an entity that stores values. There are three types of parameters:  
2293 variables (named parameters), positional parameters, and special parameters. Parameter  
2294 expansion is accomplished by introducing a parameter with the ‘\$’ character.

2295 **Note:** See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.5, Parameters and  
2296 Variables.

2297 In the C language, an object declared as part of a function declaration or definition that acquires  
2298 a value on entry to the function, or an identifier following the macro name in a function-like  
2299 macro definition.

**2300 3.263 Parent Directory**

2301 When discussing a given directory, the directory that both contains a directory entry for the  
2302 given directory and is represented by the pathname dot-dot in the given directory.

2303 When discussing other types of files, a directory containing a directory entry for the file under  
2304 discussion.

2305 This concept does not apply to dot and dot-dot.

**2306 3.264 Parent Process**

2307 The process which created (or inherited) the process under discussion.

**2308 3.265 Parent Process ID**

2309 An attribute of a new process identifying the parent of the process. The parent process ID of a  
2310 process is the process ID of its creator, for the lifetime of the creator. After the creator’s lifetime  
2311 has ended, the parent process ID is the process ID of an implementation-defined system process.

**2312 3.266 Pathname**

2313 A character string that is used to identify a file. In the context of IEEE Std 1003.1-2001, a  
2314 pathname consists of, at most, {PATH\_MAX} bytes, including the terminating null byte. It has an  
2315 optional beginning slash, followed by zero or more filenames separated by slashes. A pathname  
2316 may optionally contain one or more trailing slashes. Multiple successive slashes are considered  
2317 to be the same as one slash.

2318 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 100).

**2319 3.267 Pathname Component**

2320 See *Filename* in Section 3.169 (on page 58).

**2321 3.268 Path Prefix**

2322 A pathname, with an optional ending slash, that refers to a directory.

**2323 3.269 Pattern**

2324 A sequence of characters used either with regular expression notation or for pathname  
2325 expansion, as a means of selecting various character strings or pathnames, respectively.

2326 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 169).

2327 See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.6, Pathname  
2328 Expansion.

2329 The syntaxes of the two types of patterns are similar, but not identical; IEEE Std 1003.1-2001  
2330 always indicates the type of pattern being referred to in the immediate context of the use of the  
2331 term.

**2332 3.270 Period**

2333 The character ' . '. The term “period” is contrasted with dot (see also Section 3.135 (on page  
2334 53)), which is used to describe a specific directory entry.

**2335 3.271 Permissions**

2336 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2337 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 97).

**2338 3.272 Persistence**

2339 A mode for semaphores, shared memory, and message queues requiring that the object and its  
2340 state (including data, if any) are preserved after the object is no longer referenced by any process.

2341 Persistence of an object does not imply that the state of the object is maintained across a system  
2342 crash or a system reboot.

2343 **3.273 Pipe**

2344 An object accessed by one of the pair of file descriptors created by the *pipe()* function. Once  
 2345 created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO  
 2346 special file when accessed in this way. It has no name in the file hierarchy.

2347 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of  
 2348 IEEE Std 1003.1-2001.

2349 **3.274 Polling**

2350 A scheduling scheme whereby the local process periodically checks until the pre-specified  
 2351 events (for example, read, write) have occurred.

2352 **3.275 Portable Character Set**

2353 The collection of characters that are required to be present in all locales supported by  
 2354 conforming systems.

2355 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 113).

2356 This term is contrasted against the smaller portable filename character set; see also Section 3.276.

2357 **3.276 Portable Filename Character Set**

2358 The set of characters from which portable filenames are constructed.

2359 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  
 2360 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  
 2361 0 1 2 3 4 5 6 7 8 9 . \_ -

2362 The last three characters are the period, underscore, and hyphen characters, respectively.

2363 **3.277 Positional Parameter**

2364 In the shell command language, a parameter denoted by a single digit or one or more digits in  
 2365 curly braces.

2366 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section  
 2367 2.5.1, Positional Parameters.

2368 **3.278 Preallocation**

2369 The reservation of resources in a system for a particular use.

2370 Preallocation does not imply that the resources are immediately allocated to that use, but merely  
 2371 indicates that they are guaranteed to be available in bounded time when needed.

**2372 3.279 Preempted Process (or Thread)**

2373 A running thread whose execution is suspended due to another thread becoming runnable at a  
2374 higher priority.

**2375 3.280 Previous Job**

2376 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the  
2377 current job exits. There is at most one previous job; see also Section 3.203 (on page 63).

**2378 3.281 Printable Character**

2379 One of the characters included in the **print** character classification of the *LC\_CTYPE* category in  
2380 the current locale.

2381 **Note:** The *LC\_CTYPE* category is defined in detail in Section 7.3.1 (on page 126).

**2382 3.282 Printable File**

2383 A text file consisting only of the characters included in the **print** and **space** character  
2384 classifications of the *LC\_CTYPE* category and the <backspace>, all in the current locale.

2385 **Note:** The *LC\_CTYPE* category is defined in detail in Section 7.3.1 (on page 126).

**2386 3.283 Priority**

2387 A non-negative integer associated with processes or threads whose value is constrained to a  
2388 range defined by the applicable scheduling policy. Numerically higher values represent higher  
2389 priorities.

**2390 3.284 Priority Band**

2391 The queuing order applied to normal priority STREAMS messages. High priority STREAMS  
2392 messages are not grouped by priority bands. The only differentiation made by the STREAMS  
2393 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate  
2394 between priority bands.

**2395 3.285 Priority Inversion**

2396 A condition in which a thread that is not voluntarily suspended (waiting for an event or time  
2397 delay) is not running while a lower priority thread is running. Such blocking of the higher  
2398 priority thread is often caused by contention for a shared resource.

**2399 3.286 Priority Scheduling**

2400 A performance and determinism improvement facility to allow applications to determine the  
2401 order in which threads that are ready to run are granted access to processor resources.



**2402 3.287 Priority-Based Scheduling**

2403 Scheduling in which the selection of a running thread is determined by the priorities of the  
2404 runnable processes or threads.

**2405 3.288 Privilege**

2406 See *Appropriate Privileges* in Section 3.19 (on page 37).

**2407 3.289 Process**

2408 An address space with one or more threads executing within that address space, and the  
2409 required system resources for those threads.

2410 **Note:** Many of the system resources defined by IEEE Std 1003.1-2001 are shared among all of the  
2411 threads within a process. These include the process ID, the parent process ID, process group ID,  
2412 session membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-  
2413 ID, supplementary group IDs, current working directory, root directory, file mode creation  
2414 mask, and file descriptors.

**2415 3.290 Process Group**

2416 A collection of processes that permits the signaling of related processes. Each process in the  
2417 system is a member of a process group that is identified by a process group ID. A newly created  
2418 process joins the process group of its creator.

**2419 3.291 Process Group ID**

2420 The unique positive integer identifier representing a process group during its lifetime.

2421 **Note:** See also Process Group ID Reuse defined in Section 4.12 (on page 101).

**2422 3.292 Process Group Leader**

2423 A process whose process ID is the same as its process group ID.

**2424 3.293 Process Group Lifetime**

2425 A period of time that begins when a process group is created and ends when the last remaining  
2426 process in the group leaves the group, due either to the end of the last process' lifetime or to the  
2427 last remaining process calling the *setsid()* or *setpgid()* functions.

2428 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of  
2429 IEEE Std 1003.1-2001.

### 2430 3.294 Process ID

2431 The unique positive integer identifier representing a process during its lifetime.

2432 **Note:** See also Process ID Reuse defined in Section 4.12 (on page 101).

### 2433 3.295 Process Lifetime

2434 The period of time that begins when a process is created and ends when its process ID is  
 2435 returned to the system. After a process is created by *fork()*, *posix\_spawn()*, *posix\_spawnp()*, or 2  
 2436 *vfork()*, it is considered active. At least one thread of control and address space exist until it 2  
 2437 terminates. It then enters an inactive state where certain resources may be returned to the  
 2438 system, although some resources, such as the process ID, are still in use. When another process  
 2439 executes a *wait()*, *waitid()*, or *waitpid()* function for an inactive process, the remaining resources  
 2440 are returned to the system. The last resource to be returned to the system is the process ID. At  
 2441 this time, the lifetime of the process ends.

2442 **Note:** The *fork()*, *posix\_spawn()*, *posix\_spawnp()*, *vfork()*, *wait()*, *waitid()*, and *waitpid()* functions are 2  
 2443 defined in detail in the System Interfaces volume of IEEE Std 1003.1-2001.

### 2444 3.296 Process Memory Locking

2445 A performance improvement facility to bind application programs into the high-performance  
 2446 random access memory of a computer system. This avoids potential latencies introduced by the  
 2447 operating system in storing parts of a program that were not recently referenced on secondary  
 2448 memory devices.

### 2449 3.297 Process Termination

2450 There are two kinds of process termination:

- 2451 1. Normal termination occurs by a return from *main()*, when requested with the *exit()*, 2  
 2452 *\_exit()*, or *\_Exit()* functions; or when the last thread in the process terminates by returning 2  
 2453 from its start function, by calling the *pthread\_exit()* function, or through cancellation. 2
- 2454 2. Abnormal termination occurs when requested by the *abort()* function or when some  
 2455 signals are received.

2456 **Note:** The *\_exit()*, *\_Exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces 2  
 2457 volume of IEEE Std 1003.1-2001.

### 2458 3.298 Process-To-Process Communication

2459 The transfer of data between processes.

### 2460 3.299 Process Virtual Time

2461 The measurement of time in units elapsed by the system clock while a process is executing.

**2462 3.300 Program**

2463 A prepared sequence of instructions to the system to accomplish a defined task. The term  
2464 “program” in IEEE Std 1003.1-2001 encompasses applications written in the Shell Command  
2465 Language, complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level  
2466 languages.

**2467 3.301 Protocol**

2468 A set of semantic and syntactic rules for exchanging information.

**2469 3.302 Pseudo-Terminal**

2470 A facility that provides an interface that is identical to the terminal subsystem. A pseudo-  
2471 terminal is composed of two devices: the “master device” and a “slave device”. The slave device  
2472 provides processes with an interface that is identical to the terminal interface, although there  
2473 need not be hardware behind that interface. Anything written on the master device is presented  
2474 to the slave as an input and anything written on the slave device is presented as an input on the  
2475 master side.

**2476 3.303 Radix Character**

2477 The character that separates the integer part of a number from the fractional part.

**2478 3.304 Read-Only File System**

2479 A file system that has implementation-defined characteristics restricting modifications.

2480 **Note:** File Times Update is described in detail in Section 4.7 (on page 98).

**2481 3.305 Read-Write Lock**

2482 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous  
2483 read-only access to data while allowing only one thread to have write access at any given time.  
2484 They are typically used to protect data that is read-only more frequently than it is changed.

2485 Read-write locks can be used to synchronize threads in the current process and other processes if  
2486 they are allocated in memory that is writable and shared among the cooperating processes and  
2487 have been initialized for this behavior.

**2488 3.306 Real Group ID**

2489 The attribute of a process that, at the time of process creation, identifies the group of the user  
2490 who created the process; see also Section 3.188 (on page 61).

### 2491 **3.307 Real Time**

2492 Time measured as total units elapsed by the system clock without regard to which thread is  
2493 executing.

### 2494 **3.308 Realtime Signal Extension**

2495 A determinism improvement facility to enable asynchronous signal notifications to an  
2496 application to be queued without impacting compatibility with the existing signal functions.

### 2497 **3.309 Real User ID**

2498 The attribute of a process that, at the time of process creation, identifies the user who created the  
2499 process; see also Section 3.425 (on page 93).

### 2500 **3.310 Record**

2501 A collection of related data units or words which is treated as a unit.

### 2502 **3.311 Redirection**

2503 In the shell command language, a method of associating files with the input or output of  
2504 commands.

2505 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.7,  
2506 Redirection.

### 2507 **3.312 Redirection Operator**

2508 In the shell command language, a token that performs a redirection function. It is one of the  
2509 following symbols:

2510        <        >        > |        <<        >>        <&        >&        <<-        <>

### 2511 **3.313 Reentrant Function**

2512 A function whose effect, when called by two or more threads, is guaranteed to be as if the  
2513 threads each executed the function one after another in an undefined order, even if the actual  
2514 execution is interleaved.

### 2515 **3.314 Referenced Shared Memory Object**

2516 A shared memory object that is open or has one or more mappings defined on it.

**2517 3.315 Refresh**

2518 To ensure that the information on the user's terminal screen is up-to-date.

**2519 3.316 Regular Expression**

2520 A pattern that selects specific strings from a set of character strings.

2521 **Note:** Regular Expressions are described in detail in Chapter 9 (on page 169).

**2522 3.317 Region**

2523 In the context of the address space of a process, a sequence of addresses.

2524 In the context of a file, a sequence of offsets.

**2525 3.318 Regular File**

2526 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the  
2527 system.

**2528 3.319 Relative Pathname**

2529 A pathname not beginning with a slash.

2530 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 100).

**2531 3.320 Relocatable File**

2532 A file holding code or data suitable for linking with other object files to create an executable or a  
2533 shared object file.

**2534 3.321 Relocation**

2535 The process of connecting symbolic references with symbolic definitions. For example, when a  
2536 program calls a function, the associated call instruction transfers control to the proper  
2537 destination address at execution.

**2538 3.322 Requested Batch Service**

2539 A service that is either rejected or performed prior to a response from the service to the  
2540 requester.

**2541 3.323 (Time) Resolution**

2542 The minimum time interval that a clock can measure or whose passage a timer can detect.

**2543 3.324 Root Directory**

2544 A directory, associated with a process, that is used in pathname resolution for pathnames that  
2545 begin with a slash.

**2546 3.325 Runnable Process (or Thread)**

2547 A thread that is capable of being a running thread, but for which no processor is available.

**2548 3.326 Running Process (or Thread)**

2549 A thread currently executing on a processor. On multi-processor systems there may be more  
2550 than one such thread in a system at a time.

**2551 3.327 Saved Resource Limits**

2552 An attribute of a process that provides some flexibility in the handling of unrepresentable  
2553 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2554 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of  
2555 IEEE Std 1003.1-2001.

**2556 3.328 Saved Set-Group-ID**

2557 An attribute of a process that allows some flexibility in the assignment of the effective group ID  
2558 attribute, as described in the *exec* family of functions and *setgid()*.

2559 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of  
2560 IEEE Std 1003.1-2001.

**2561 3.329 Saved Set-User-ID**

2562 An attribute of a process that allows some flexibility in the assignment of the effective user ID  
2563 attribute, as described in the *exec* family of functions and *setuid()*.

2564 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of  
2565 IEEE Std 1003.1-2001.

**2566 3.330 Scheduling**

2567 The application of a policy to select a runnable process or thread to become a running process or  
2568 thread, or to alter one or more of the thread lists.

**2569 3.331 Scheduling Allocation Domain**

2570 The set of processors on which an individual thread can be scheduled at any given time.

### 2571 **3.332 Scheduling Contention Scope**

2572 A property of a thread that defines the set of threads against which that thread competes for  
2573 resources.

2574 For example, in a scheduling decision, threads sharing scheduling contention scope compete for  
2575 processor resources. In IEEE Std 1003.1-2001, a thread has scheduling contention scope of either  
2576 PTHREAD\_SCOPE\_SYSTEM or PTHREAD\_SCOPE\_PROCESS.

### 2577 **3.333 Scheduling Policy**

2578 A set of rules that is used to determine the order of execution of processes or threads to achieve  
2579 some goal.

2580 **Note:** Scheduling Policy is defined in detail in Section 4.13 (on page 101).

### 2581 **3.334 Screen**

2582 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a  
2583 physical display device or may occupy the entire physical area of the display device.

### 2584 **3.335 Scroll**

2585 To move the representation of data vertically or horizontally relative to the terminal screen.  
2586 There are two types of scrolling:

- 2587 1. The cursor moves with the data.
- 2588 2. The cursor remains stationary while the data moves.

### 2589 **3.336 Semaphore**

2590 A minimum synchronization primitive to serve as a basis for more complex synchronization  
2591 mechanisms to be defined by the application program.

2592 **Note:** Semaphores are defined in detail in Section 4.15 (on page 102).

### 2593 **3.337 Session**

2594 A collection of process groups established for job control purposes. Each process group is a  
2595 member of a session. A process is considered to be a member of the session of which its process  
2596 group is a member. A newly created process joins the session of its creator. A process can alter  
2597 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2598 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of  
2599 IEEE Std 1003.1-2001.

**2600 3.338 Session Leader**

2601 A process that has created a session.

2602 **Note:** For further information, see the *setuid()* function defined in the System Interfaces volume of  
2603 IEEE Std 1003.1-2001.

**2604 3.339 Session Lifetime**

2605 The period between when a session is created and the end of the lifetime of all the process  
2606 groups that remain as members of the session.

**2607 3.340 Shared Memory Object**

2608 An object that represents memory that can be mapped concurrently into the address space of  
2609 more than one process.

**2610 3.341 Shell**

2611 A program that interprets sequences of text input as commands. It may operate on an input  
2612 stream or it may interactively prompt and read commands from a terminal.

**2613 3.342 Shell, the**

2614 The Shell Command Language Interpreter; a specific instance of a shell.

2615 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of  
2616 IEEE Std 1003.1-2001.

**2617 3.343 Shell Script**

2618 A file containing shell commands. If the file is made executable, it can be executed by specifying  
2619 its name as a simple command. Execution of a shell script causes a shell to execute the  
2620 commands within the script. Alternatively, a shell can be requested to execute the commands in  
2621 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2622 **Note:** Simple Commands are defined in detail in the Shell and Utilities volume of  
2623 IEEE Std 1003.1-2001, Section 2.9.1, Simple Commands.

2624 The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

**2625 3.344 Signal**

2626 A mechanism by which a process or thread may be notified of, or affected by, an event occurring  
2627 in the system. Examples of such events include hardware exceptions and specific actions by  
2628 processes. The term signal is also used to refer to the event itself.



**2629 3.345 Signal Stack**

2630 Memory established for a thread, in which signal handlers catching signals sent to that thread  
2631 are executed.

**2632 3.346 Single-Quote**

2633 The character ' ' , also known as apostrophe.

**2634 3.347 Slash**

2635 The character ' / ' , also known as solidus.

**2636 3.348 Socket**

2637 A file of a particular type that is used as a communications endpoint for process-to-process  
2638 communication as described in the System Interfaces volume of IEEE Std 1003.1-2001.

**2639 3.349 Socket Address**

2640 An address associated with a socket or remote endpoint, including an address family identifier  
2641 and addressing information specific to that address family. The address may include multiple  
2642 parts, such as a network address associated with a host system and an identifier for a specific  
2643 endpoint.

**2644 3.350 Soft Limit**

2645 A resource limitation established for each process that the process may set to any value less than  
2646 or equal to the hard limit.

**2647 3.351 Source Code**

2648 When dealing with the Shell Command Language, input to the command language interpreter.  
2649 The term “shell script” is synonymous with this meaning.

2650 When dealing with an ISO/IEC-conforming programming language, source code is input to a  
2651 compiler conforming to that ISO/IEC standard.

2652 Source code also refers to the input statements prepared for the following standard utilities:  
2653 *awk, bc, ed, lex, localedef, make, sed, and yacc.*

2654 Source code can also refer to a collection of sources meeting any or all of these meanings.

2655 **Note:** The *awk, bc, ed, lex, localedef, make, sed, and yacc* utilities are defined in detail in the Shell and  
2656 Utilities volume of IEEE Std 1003.1-2001.

2657 **3.352 Space Character (<space>)**

2658 The character defined in the portable character set as <space>. The <space> is a member of the  
 2659 **space** character class of the current locale, but represents the single character, and not all of the  
 2660 possible members of the class; see also Section 3.431 (on page 94).

2661 **3.353 Spawn**

2662 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient  
 2663 replacement for *fork()/exec*.

2664 **3.354 Special Built-In**

2665 See *Built-In Utility* in Section 3.83 (on page 46).

2666 **3.355 Special Parameter**

2667 In the shell command language, a parameter named by a single character from the following list:

2668 \* @ # ? ! - \$ 0

2669 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section  
 2670 2.5.2, Special Parameters.

2671 **3.356 Spin Lock**

2672 A synchronization object used to allow multiple threads to serialize their access to shared data.

2673 **3.357 Sporadic Server**

2674 A scheduling policy for threads and processes that reserves a certain amount of execution  
 2675 capacity for processing aperiodic events at a given priority level.

2676 **3.358 Standard Error**

2677 An output stream usually intended to be used for diagnostic messages.

2678 **3.359 Standard Input**

2679 An input stream usually intended to be used for primary data input.

2680 **3.360 Standard Output**

2681 An output stream usually intended to be used for primary data output.

### 2682 **3.361 Standard Utilities**

2683 The utilities described in the Shell and Utilities volume of IEEE Std 1003.1-2001.

### 2684 **3.362 Stream**

2685 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence  
 2686 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,  
 2687 *fopen()*, or *popen()* functions, and are associated with a file descriptor. A stream provides the  
 2688 additional services of user-selectable buffering and formatted input and output; see also Section  
 2689 3.363.

2690 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.5,  
 2691 Standard I/O Streams.

2692 The *fdopen()*, *fopen()*, or *popen()* functions are defined in detail in the System Interfaces volume  
 2693 of IEEE Std 1003.1-2001.

### 2694 **3.363 STREAM**

2695 Appearing in uppercase, STREAM refers to a full-duplex connection between a process and an  
 2696 open device or pseudo-device. It optionally includes one or more intermediate processing  
 2697 modules that are interposed between the process end of the STREAM and the device driver (or  
 2698 pseudo-device driver) end of the STREAM; see also Section 3.362.

2699 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.6,  
 2700 STREAMS.

### 2701 **3.364 STREAM End**

2702 The STREAM end is the driver end of the STREAM and is also known as the downstream end of  
 2703 the STREAM.

### 2704 **3.365 STREAM Head**

2705 The STREAM head is the beginning of the STREAM and is at the boundary between the system  
 2706 and the application process. This is also known as the upstream end of the STREAM.

### 2707 **3.366 STREAMS Multiplexor**

2708 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected above  
 2709 is referred to as N-to-1, or “upper multiplexing”. Multiplexing with STREAMS connected below  
 2710 is referred to as 1-to-N or “lower multiplexing”.

### 2711 **3.367 String**

2712 A contiguous sequence of bytes terminated by and including the first null byte.

**2713 3.368 Subshell**

2714 A shell execution environment, distinguished from the main or current shell execution  
2715 environment.

2716 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.12,  
2717 Shell Execution Environment.

**2718 3.369 Successfully Transferred**

2719 For a write operation to a regular file, when the system ensures that all data written is readable  
2720 on any subsequent open of the file (even one that follows a system or power failure) in the  
2721 absence of a failure of the physical storage medium.

2722 For a read operation, when an image of the data on the physical storage medium is available to  
2723 the requesting process.

**2724 3.370 Supplementary Group ID**

2725 An attribute of a process used in determining file access permissions. A process has up to  
2726 {NGROUPS\_MAX} supplementary group IDs in addition to the effective group ID. The  
2727 supplementary group IDs of a process are set to the supplementary group IDs of the parent  
2728 process when the process is created.

**2729 3.371 Suspended Job**

2730 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the  
2731 process group to stop. A suspended job is a background job, but a background job is not  
2732 necessarily a suspended job.

**2733 3.372 Symbolic Link**

2734 A type of file with the property that when the file is encountered during pathname resolution, a  
2735 string stored by the file is used to modify the pathname resolution. The stored string has a length  
2736 of {SYMLINK\_MAX} bytes or fewer.

2737 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 100).

**2738 3.373 Synchronized Input and Output**

2739 A determinism and robustness improvement mechanism to enhance the data input and output  
2740 mechanisms, so that an application can ensure that the data being manipulated is physically  
2741 present on secondary mass storage devices.

**2742 3.374 Synchronized I/O Completion**

2743 The state of an I/O operation that has either been successfully transferred or diagnosed as  
2744 unsuccessful.

**2745 3.375 Synchronized I/O Data Integrity Completion**

2746 For read, when the operation has been completed or diagnosed if unsuccessful. The read is  
2747 complete only when an image of the data has been successfully transferred to the requesting  
2748 process. If there were any pending write requests affecting the data to be read at the time that  
2749 the synchronized read operation was requested, these write requests are successfully transferred  
2750 prior to reading the data.

2751 For write, when the operation has been completed or diagnosed if unsuccessful. The write is  
2752 complete only when the data specified in the write request is successfully transferred and all file  
2753 system information required to retrieve the data is successfully transferred.

2754 File attributes that are not necessary for data retrieval (access time, modification time, status  
2755 change time) need not be successfully transferred prior to returning to the calling process.

**2756 3.376 Synchronized I/O File Integrity Completion**

2757 Identical to a synchronized I/O data integrity completion with the addition that all file attributes  
2758 relative to the I/O operation (including access time, modification time, status change time) are  
2759 successfully transferred prior to returning to the calling process.

**2760 3.377 Synchronized I/O Operation**

2761 An I/O operation performed on a file that provides the application assurance of the integrity of  
2762 its data and files.

**2763 3.378 Synchronous I/O Operation**

2764 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the  
2765 processor until that I/O operation completes.

2766 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or  
2767 synchronized I/O file integrity completion.

**2768 3.379 Synchronously-Generated Signal**

2769 A signal that is attributable to a specific thread.

2770 For example, a thread executing an illegal instruction or touching invalid memory causes a  
2771 synchronously-generated signal. Being synchronous is a property of how the signal was  
2772 generated and not a property of the signal number.

**2773 3.380 System**

2774 An implementation of IEEE Std 1003.1-2001.

### 2775 3.381 System Crash

2776 An interval initiated by an unspecified circumstance that causes all processes (possibly other  
 2777 than special system processes) to be terminated in an undefined manner, after which any  
 2778 changes to the state and contents of files created or written to by an application prior to the  
 2779 interval are undefined, except as required elsewhere in IEEE Std 1003.1-2001.

### 2780 3.382 System Console

2781 A device that receives messages sent by the *syslog()* function, and the *fmtmsg()* function when 2  
 2782 the MM\_CONSOLE flag is set. 2

2783 **Note:** The *syslog()* and *fmtmsg()* functions are defined in detail in the System Interfaces volume of  
 2784 IEEE Std 1003.1-2001.

### 2785 3.383 System Databases

2786 An implementation provides two system databases: the “group database” (see also Section 3.187 2  
 2787 (on page 61)) and the “user database” (see also Section 3.424 (on page 93)). 2

### 2788 3.384 System Documentation

2789 All documentation provided with an implementation except for the conformance document.  
 2790 Electronically distributed documents for an implementation are considered part of the system  
 2791 documentation.

### 2792 3.385 System Process

2793 An object other than a process executing an application, that is provided by the system and has a 2  
 2794 process ID. 2

### 2795 3.386 System Reboot

2796 An unspecified sequence of events that may result in the loss of transitory data; that is, data that 2  
 2797 is not saved in permanent storage. For example, message queues, shared memory, semaphores,  
 2798 and processes.

### 2799 3.387 System Trace Event

2800 A trace event that is generated by the implementation, in response either to a system-initiated  
 2801 action or to an application-requested action, except for a call to *posix\_trace\_event()*. When  
 2802 supported by the implementation, a system-initiated action generates a process-independent  
 2803 system trace event and an application-requested action generates a process-dependent system  
 2804 trace event. For a system trace event not defined by IEEE Std 1003.1-2001, the associated trace  
 2805 event type identifier is derived from the implementation-defined name for this trace event, and  
 2806 the associated data is of implementation-defined content and length.

**2807 3.388 System-Wide**

2808 Pertaining to events occurring in all processes existing in an implementation at a given point in  
2809 time.

**2810 3.389 Tab Character (<tab>)**

2811 A character that in the output stream indicates that printing or displaying should start at the  
2812 next horizontal tabulation position on the current line. It is the character designated by ‘\t’ in  
2813 the C language. If the current position is at or past the last defined horizontal tabulation  
2814 position, the behavior is unspecified. It is unspecified whether this character is the exact  
2815 sequence transmitted to an output device by the system to accomplish the tabulation.

**2816 3.390 Terminal (or Terminal Device)**

2817 A character special file that obeys the specifications of the general terminal interface.

2818 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 187).

**2819 3.391 Text Column**

2820 A roughly rectangular block of characters capable of being laid out side-by-side next to other  
2821 text columns on an output page or terminal screen. The widths of text columns are measured in  
2822 column positions.

**2823 3.392 Text File**

2824 A file that contains characters organized into one or more lines. The lines do not contain NUL  
2825 characters and none can exceed {LINE\_MAX} bytes in length, including the <newline>.  
2826 Although IEEE Std 1003.1-2001 does not distinguish between text files and binary files (see the  
2827 ISO C standard), many utilities only produce predictable or meaningful output when operating  
2828 on text files. The standard utilities that have such restrictions always specify “text files” in their  
2829 STDIN or INPUT FILES sections.

**2830 3.393 Thread**

2831 A single flow of control within a process. Each thread has its own thread ID, scheduling priority  
2832 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to  
2833 support a flow of control. Anything whose address may be determined by a thread, including  
2834 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage  
2835 obtained through implementation-defined functions, and automatic variables, are accessible to  
2836 all threads in the same process.

2837 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of  
2838 IEEE Std 1003.1-2001.

2839 **3.394 Thread ID**

2840 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread\_t**  
 2841 called a thread ID.

2842 **3.395 Thread List**

2843 An ordered set of runnable threads that all have the same ordinal value for their priority.

2844 The ordering of threads on the list is determined by a scheduling policy or policies. The set of  
 2845 thread lists includes all runnable threads in the system.

2846 **3.396 Thread-Safe**

2847 A function that may be safely invoked concurrently by multiple threads. Each function defined  
 2848 in the System Interfaces volume of IEEE Std 1003.1-2001 is thread-safe unless explicitly stated  
 2849 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is  
 2850 accessing static storage, or objects shared among threads.

2851 **3.397 Thread-Specific Data Key**

2852 A process global handle of type **pthread\_key\_t** which is used for naming thread-specific data.

2853 Although the same key value may be used by different threads, the values bound to the key by  
 2854 *pthread\_setspecific()* and accessed by *pthread\_getspecific()* are maintained on a per-thread basis  
 2855 and persist for the life of the calling thread.

2856 **Note:** The *pthread\_getspecific()* and *pthread\_setspecific()* functions are defined in detail in the System  
 2857 Interfaces volume of IEEE Std 1003.1-2001.

2858 **3.398 Tilde**

2859 The character ‘~’.

2860 **3.399 Timeouts**

2861 A method of limiting the length of time an interface will block; see also Section 3.76 (on page 45).

2862 **3.400 Timer**

2863 A mechanism that can notify a thread when the time as measured by a particular clock has  
 2864 reached or passed a specified value, or when a specified amount of time has passed.

2865 **3.401 Timer Overrun**

2866 A condition that occurs each time a timer, for which there is already an expiration signal queued  
 2867 to the process, expires.



**2868 3.402 Token**

2869 In the shell command language, a sequence of characters that the shell considers as a single unit  
2870 when reading input. A token is either an operator or a word.

2871 **Note:** The rules for reading input are defined in detail in the Shell and Utilities volume of  
2872 IEEE Std 1003.1-2001, Section 2.3, Token Recognition.

**2873 3.403 Trace Analyzer Process**

2874 A process that extracts trace events from a trace stream to retrieve information about the  
2875 behavior of an application.

**2876 3.404 Trace Controller Process**

2877 A process that creates a trace stream for tracing a process.

**2878 3.405 Trace Event**

2879 A data object that represents an action executed by the system, and that is recorded in a trace  
2880 stream.

**2881 3.406 Trace Event Type**

2882 A data object type that defines a class of trace event.

**2883 3.407 Trace Event Type Mapping**

2884 A one-to-one mapping between trace event types and trace event names.

**2885 3.408 Trace Filter**

2886 A filter that allows the trace controller process to specify those trace event types that are to be  
2887 ignored; that is, not generated.

**2888 3.409 Trace Generation Version**

2889 A data object that is an implementation-defined character string, generated by the trace system  
2890 and describing the origin and version of the trace system.

**2891 3.410 Trace Log**

2892 The flushed image of a trace stream, if the trace stream is created with a trace log.

**2893 3.411 Trace Point**

2894 An action that may cause a trace event to be generated.

**2895 3.412 Trace Stream**

2896 An opaque object that contains trace events plus internal data needed to interpret those trace  
2897 events.

**2898 3.413 Trace Stream Identifier**

2899 A handle to manage tracing operations in a trace stream.

**2900 3.414 Trace System**

2901 A system that allows both system and user trace events to be generated into a trace stream.  
2902 These trace events can be retrieved later.

**2903 3.415 Traced Process**

2904 A process for which at least one trace stream has been created. A traced process is also called a  
2905 target process.

**2906 3.416 Tracing Status of a Trace Stream**

2907 A status that describes the state of an active trace stream. The tracing status of a trace stream can  
2908 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:  
2909 running or suspended.

**2910 3.417 Typed Memory Name Space**

2911 A system-wide name space that contains the names of the typed memory objects present in the  
2912 system. It is configurable for a given implementation.

**2913 3.418 Typed Memory Object**

2914 A combination of a typed memory pool and a typed memory port. The entire contents of the  
2915 pool are accessible from the port. The typed memory object is identified through a name that  
2916 belongs to the typed memory name space.

**2917 3.419 Typed Memory Pool**

2918 An extent of memory with the same operational characteristics. Typed memory pools may be  
2919 contained within each other.

### 2920 **3.420 Typed Memory Port**

2921 A hardware access path to one or more typed memory pools.

### 2922 **3.421 Unbind**

2923 Remove the association between a network address and an endpoint.

### 2924 **3.422 Unit Data**

2925 See *Datagram* in Section 3.123 (on page 52).

### 2926 **3.423 Upshifting**

2927 The conversion of a lowercase character that has a single-character uppercase representation  
2928 into this uppercase representation.

### 2929 **3.424 User Database**

2930 A system database that contains at least the following information for each user ID: 2

- 2931 • User name
- 2932 • Numerical user ID
- 2933 • Initial numerical group ID
- 2934 • Initial working directory
- 2935 • Initial user program

2936 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under  
2937 which the initial values are operative are implementation-defined.

2938 If the initial user program field is null, an implementation-defined program is used.

2939 If the initial working directory field is null, the interpretation of that field is implementation-  
2940 defined.

2941 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

### 2942 **3.425 User ID**

2943 A non-negative integer that is used to identify a system user. When the identity of a user is  
2944 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a  
2945 saved set-user-ID.

### 2946 **3.426 User Name**

2947 A string that is used to identify a user; see also Section 3.424. To be portable across systems  
2948 conforming to IEEE Std 1003.1-2001, the value is composed of characters from the portable  
2949 filename character set. The hyphen should not be used as the first character of a portable user

2950 name.

### 2951 **3.427 User Trace Event**

2952 A trace event that is generated explicitly by the application as a result of a call to  
2953 *posix\_trace\_event()*.

### 2954 **3.428 Utility**

2955 A program, excluding special built-in utilities provided as part of the Shell Command Language,  
2956 that can be called by name from a shell to perform a specific task, or related set of tasks.

2957 **Note:** For further information on special built-in utilities, see the Shell and Utilities volume of  
2958 IEEE Std 1003.1-2001, Section 2.14, Special Built-In Utilities.

### 2959 **3.429 Variable**

2960 In the shell command language, a named parameter.

2961 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.5,  
2962 Parameters and Variables.

### 2963 **3.430 Vertical-Tab Character (<vertical-tab>)**

2964 A character that in the output stream indicates that printing should start at the next vertical  
2965 tabulation position. It is the character designated by '`\v`' in the C language. If the current  
2966 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is  
2967 unspecified whether this character is the exact sequence transmitted to an output device by the  
2968 system to accomplish the tabulation.

### 2969 **3.431 White Space**

2970 A sequence of one or more characters that belong to the **space** character class as defined via the  
2971 *LC\_CTYPE* category in the current locale.

2972 In the POSIX locale, white space consists of one or more <blank>s (<space>s and <tab>s),  
2973 <newline>s, <carriage-return>s, <form-feed>s, and <vertical-tab>s.

### 2974 **3.432 Wide-Character Code (C Language)**

2975 An integer value corresponding to a single graphic symbol or control code.

2976 **Note:** C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 117).

### 2977 **3.433 Wide-Character Input/Output Functions**

2978 The functions that perform wide-oriented input from streams or wide-oriented output to  
2979 streams: *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *putwc()*,

2980 *putwchar()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vwprintf()*, *vwscanf()*, *wprintf()*, and *wscanf()*.

2981 **Note:** These functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-2001.

### 2982 **3.434 Wide-Character String**

2983 A contiguous sequence of wide-character codes terminated by and including the first null wide-  
2984 character code.

### 2985 **3.435 Word**

2986 In the shell command language, a token other than an operator. In some cases a word is also a  
2987 portion of a word token: in the various forms of parameter expansion, such as *\${name-word}*, and  
2988 variable assignment, such as *name=word*, the word is the portion of the token depicted by *word*.  
2989 The concept of a word is no longer applicable following word expansions—only fields remain.

2990 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section  
2991 2.6.2, Parameter Expansion and the Shell and Utilities volume of IEEE Std 1003.1-2001, Section  
2992 2.6, Word Expansions.

### 2993 **3.436 Working Directory (or Current Working Directory)**

2994 A directory, associated with a process, that is used in pathname resolution for pathnames that  
2995 do not begin with a slash.

### 2996 **3.437 Worldwide Portability Interface**

2997 Functions for handling characters in a codeset-independent manner.

### 2998 **3.438 Write**

2999 To output characters to a file, such as standard output or standard error. Unless otherwise  
3000 stated, standard output is the default output destination for all uses of the term “write”; see the  
3001 distinction between display and write in Section 3.132 (on page 53).

### 3002 **3.439 XSI**

3003 The X/Open System Interface is the core application programming interface for C and *sh*  
3004 programming for systems conforming to the Single UNIX Specification. This is a superset of the  
3005 mandatory requirements for conformance to IEEE Std 1003.1-2001.

### 3006 **3.440 XSI-Conformant**

3007 A system which allows an application to be built using a set of services that are consistent across  
3008 all systems that conform to IEEE Std 1003.1-2001 and that support the XSI extension.

3009 **Note:** See also Chapter 2 (on page 17).

3010 **3.441 Zombie Process**

3011 A process that has terminated and that is deleted when its exit status has been reported to  
3012 another process which is waiting for that process to terminate.

3013 **3.442  $\pm 0$** 

3014 The algebraic sign provides additional information about any variable that has the value zero  
3015 when the representation allows the sign to be determined.

# General Concepts

3016

3017 For the purposes of IEEE Std 1003.1-2001, the general concepts given in Chapter 4 apply.

3018 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and  
 3019 definitions given in this chapter are used elsewhere in text related to extensions and options,  
 3020 they are shaded as appropriate.

## 3021 4.1 Concurrent Execution

3022 Functions that suspend the execution of the calling thread shall not cause the execution of other  
 3023 threads to be indefinitely suspended.

## 3024 4.2 Directory Protection

3025 If a directory is writable and the mode bit S\_ISVTX is set on the directory, a process may remove  
 3026 or rename files within that directory only if one or more of the following is true:

- 3027 • The effective user ID of the process is the same as that of the owner ID of the file.
- 3028 • The effective user ID of the process is the same as that of the owner ID of the directory.
- 3029 • The process has appropriate privileges.

3030 If the S\_ISVTX bit is set on a non-directory file, the behavior is unspecified.

## 3031 4.3 Extended Security Controls

3032 An implementation may provide implementation-defined extended security controls (see  
 3033 Section 3.159 (on page 57)). These permit an implementation to provide security mechanisms to  
 3034 implement different security policies than those described in IEEE Std 1003.1-2001. These  
 3035 mechanisms shall not alter or override the defined semantics of any of the interfaces in  
 3036 IEEE Std 1003.1-2001.

## 3037 4.4 File Access Permissions

3038 The standard file access control mechanism uses the file permission bits, as described below.

3039 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An  
 3040 additional access control mechanism shall only further restrict the access permissions defined by  
 3041 the file permission bits. An alternate file access control mechanism shall:

- 3042 • Specify file permission bits for the file owner class, file group class, and file other class of that  
 3043 file, corresponding to the access permissions.
- 3044 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the  
 3045 appropriate privilege.
- 3046 • Be disabled for a file after the file permission bits are changed for that file with *chmod()*. The  
 3047 disabling of the alternate mechanism need not disable any additional mechanisms supported

3048 by an implementation.

3049 Whenever a process requests file access permission for read, write, or execute/search, if no  
3050 additional mechanism denies access, access shall be determined as follows:

- 3051 • If a process has the appropriate privilege:
  - 3052 — If read, write, or directory search permission is requested, access shall be granted.
  - 3053 — If execute permission is requested, access shall be granted if execute permission is  
3054 granted to at least one user by the file permission bits or by an alternate access control  
3055 mechanism; otherwise, access shall be denied.
- 3056 • Otherwise:
  - 3057 — The file permission bits of a file contain read, write, and execute/search permissions for  
3058 the file owner class, file group class, and file other class.
  - 3059 — Access shall be granted if an alternate access control mechanism is not enabled and the  
3060 requested access permission bit is set for the class (file owner class, file group class, or file  
3061 other class) to which the process belongs, or if an alternate access control mechanism is  
3062 enabled and it allows the requested access; otherwise, access shall be denied.

## 3063 4.5 File Hierarchy

3064 Files in the system are organized in a hierarchical structure in which all of the non-terminal  
3065 nodes are directories and all of the terminal nodes are any other type of file. Since multiple  
3066 directory entries may refer to the same file, the hierarchy is properly described as a “directed  
3067 graph”.

## 3068 4.6 Filenames

3069 For a filename to be portable across implementations conforming to IEEE Std 1003.1-2001, it  
3070 shall consist only of the portable filename character set as defined in Section 3.276 (on page 73).

3071 The hyphen character shall not be used as the first character of a portable filename. Uppercase  
3072 and lowercase letters shall retain their unique identities between conforming implementations.  
3073 In the case of a portable pathname, the slash character may also be used.

## 3074 4.7 File Times Update

3075 Each file has three distinct associated time values: *st\_atime*, *st\_mtime*, and *st\_ctime*. The *st\_atime*  
3076 field is associated with the times that the file data is accessed; *st\_mtime* is associated with the  
3077 times that the file data is modified; and *st\_ctime* is associated with the times that the file status is  
3078 changed. These values are returned in the file characteristics structure, as described in  
3079 `<sys/stat.h>`.

3080 Each function or utility in IEEE Std 1003.1-2001 that reads or writes data or changes file status  
3081 indicates which of the appropriate time-related fields shall be “marked for update”. If an  
3082 implementation of such a function or utility marks for update a time-related field not specified  
3083 by IEEE Std 1003.1-2001, this shall be documented, except that any changes caused by pathname  
3084 resolution need not be documented. For the other functions or utilities in IEEE Std 1003.1-2001  
3085 (those that are not explicitly required to read or write file data or change file status, but that in  
3086 some implementations happen to do so), the effect is unspecified.



3087 An implementation may update fields that are marked for update immediately, or it may update  
 3088 such fields periodically. At an update point in time, any marked fields shall be set to the current  
 3089 time and the update marks shall be cleared. All fields that are marked for update shall be  
 3090 updated when the file ceases to be open by any process, or when a *stat()*, *fstat()*, or *lstat()* is  
 3091 performed on the file. Other times at which updates are done are unspecified. Marks for update,  
 3092 and updates themselves, are not done for files on read-only file systems; see Section 3.304 (on  
 3093 page 77).

## 3094 4.8 Host and Network Byte Orders

3095 When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned  
 3096 values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be  
 3097 stored in several octets. The convention is that all such values are stored with 8 bits in each octet,  
 3098 and with the first (lowest-addressed) octet holding the most-significant bits. This is called  
 3099 “network byte order”.

3100 Network byte order may not be convenient for processing actual values. For this, it is more  
 3101 sensible for values to be stored as ordinary integers. This is known as “host byte order”. In host  
 3102 byte order:

- 3103 • The most significant bit might not be stored in the first byte in address order.
- 3104 • Bits might not be allocated to bytes in any obvious order at all.

3105 8-bit values stored in **uint8\_t** objects do not require conversion to or from host byte order, as  
 3106 they have the same representation. 16 and 32-bit values can be converted using the *htonl()*,  
 3107 *htons()*, *ntohl()*, and *ntohs()* functions. When reading data that is to be converted to host byte  
 3108 order, it should either be received directly into a **uint16\_t** or **uint32\_t** object or should be copied  
 3109 from an array of bytes using *memcpy()* or similar. Passing the data through other types could  
 3110 cause the byte order to be changed. Similar considerations apply when sending data.

## 3111 4.9 Measurement of Execution Time

3112 The mechanism used to measure execution time shall be implementation-defined. The  
 3113 implementation shall also define to whom the CPU time that is consumed by interrupt handlers  
 3114 and system services on behalf of the operating system will be charged. See Section 3.117 (on  
 3115 page 51).

## 3116 4.10 Memory Synchronization

3117 Applications shall ensure that access to any memory location by more than one thread of control  
 3118 (threads or processes) is restricted such that no thread of control can read or modify a memory  
 3119 location while another thread of control may be modifying it. Such access is restricted using  
 3120 functions that synchronize thread execution and also synchronize memory with respect to other  
 3121 threads. The following functions synchronize memory with respect to other threads:

3122	<i>fork()</i>	<i>pthread_mutex_timedlock()</i>	<i>pthread_rwlock_tryrdlock()</i>
3123	<i>pthread_barrier_wait()</i>	<i>pthread_mutex_trylock()</i>	<i>pthread_rwlock_trywrlock()</i>
3124	<i>pthread_cond_broadcast()</i>	<i>pthread_mutex_unlock()</i>	<i>pthread_rwlock_unlock()</i>
3125	<i>pthread_cond_signal()</i>	<i>pthread_spin_lock()</i>	<i>pthread_rwlock_wrlock()</i>
3126	<i>pthread_cond_timedwait()</i>	<i>pthread_spin_trylock()</i>	<i>sem_post()</i>
3127	<i>pthread_cond_wait()</i>	<i>pthread_spin_unlock()</i>	<i>sem_trywait()</i>
3128	<i>pthread_create()</i>	<i>pthread_rwlock_rdlock()</i>	<i>sem_wait()</i>
3129	<i>pthread_join()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>wait()</i>
3130	<i>pthread_mutex_lock()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>waitpid()</i>

3131 The *pthread\_once()* function shall synchronize memory for the first call in each thread for a given 1  
 3132 **pthread\_once\_t** object. 1

3133 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified  
 3134 whether the invocation causes memory to be synchronized.

3135 Applications may allow more than one thread of control to read a memory location  
 3136 simultaneously.

## 3137 4.11 Pathname Resolution

3138 Pathname resolution is performed for a process to resolve a pathname to a particular file in a file  
 3139 hierarchy. There may be multiple pathnames that resolve to the same file.

3140 Each filename in the pathname is located in the directory specified by its predecessor (for  
 3141 example, in the pathname fragment **a/b**, file **b** is located in directory **a**). Pathname resolution  
 3142 shall fail if this cannot be accomplished. If the pathname begins with a slash, the predecessor of  
 3143 the first filename in the pathname shall be taken to be the root directory of the process (such  
 3144 pathnames are referred to as “absolute pathnames”). If the pathname does not begin with a  
 3145 slash, the predecessor of the first filename of the pathname shall be taken to be the current  
 3146 working directory of the process (such pathnames are referred to as “relative pathnames”).

3147 The interpretation of a pathname component is dependent on the value of {NAME\_MAX} and  
 3148 \_POSIX\_NO\_TRUNC associated with the path prefix of that component. If any pathname  
 3149 component is longer than {NAME\_MAX}, the implementation shall consider this an error.

3150 A pathname that contains at least one non-slash character and that ends with one or more  
 3151 trailing slashes shall be resolved as if a single dot character (‘.’) were appended to the  
 3152 pathname.

3153 If a symbolic link is encountered during pathname resolution, the behavior shall depend on  
 3154 whether the pathname component is at the end of the pathname and on the function being  
 3155 performed. If all of the following are true, then pathname resolution is complete:

- 3156 1. This is the last pathname component of the pathname.
- 3157 2. The pathname has no trailing slash.

- 3158           3. The function is required to act on the symbolic link itself, or certain arguments direct that  
3159           the function act on the symbolic link itself.

3160           In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the  
3161           symbolic link. If the combined length exceeds {PATH\_MAX}, and the implementation considers  
3162           this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be  
3163           returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created.  
3164           If the resulting pathname does not begin with a slash, the predecessor of the first filename of the  
3165           pathname is taken to be the directory containing the symbolic link.

3166           If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and  
3167           return an error indication. The same may happen if during the resolution process more symbolic  
3168           links were followed than the implementation allows. This implementation-defined limit shall  
3169           not be smaller than {SYMLOOP\_MAX}.

3170           The special filename dot shall refer to the directory specified by its predecessor. The special  
3171           filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case,  
3172           in the root directory, dot-dot may refer to the root directory itself.

3173           A pathname consisting of a single slash shall resolve to the root directory of the process. A null  
3174           pathname shall not be successfully resolved. A pathname that begins with two successive  
3175           slashes may be interpreted in an implementation-defined manner, although more than two  
3176           leading slashes shall be treated as a single slash.

## 3177 **4.12 Process ID Reuse**

3178           A process group ID shall not be reused by the system until the process group lifetime ends.

3179           A process ID shall not be reused by the system until the process lifetime ends. In addition, if  
3180           there exists a process group whose process group ID is equal to that process ID, the process ID  
3181           shall not be reused by the system until the process group lifetime ends. A process that is not a  
3182           system process shall not have a process ID of 1.

## 3183 **4.13 Scheduling Policy**

3184           A scheduling policy affects process or thread ordering:

- 3185           • When a process or thread is a running thread and it becomes a blocked thread
- 3186           • When a process or thread is a running thread and it becomes a preempted thread
- 3187           • When a process or thread is a blocked thread and it becomes a runnable thread
- 3188           • When a running thread calls a function that can change the priority or scheduling policy of a  
3189           process or thread
- 3190           • In other scheduling policy-defined circumstances

3191           Conforming implementations shall define the manner in which each of the scheduling policies  
3192           may modify the priorities or otherwise affect the ordering of processes or threads at each of the  
3193           occurrences listed above. Additionally, conforming implementations shall define in what other  
3194           circumstances and in what manner each scheduling policy may modify the priorities or affect  
3195           the ordering of processes or threads.

## 3196 4.14 Seconds Since the Epoch

3197 A value that approximates the number of seconds that have elapsed since the Epoch. A  
 3198 Coordinated Universal Time name (specified in terms of seconds (*tm\_sec*), minutes (*tm\_min*),  
 3199 hours (*tm\_hour*), days since January 1 of the year (*tm\_yday*), and calendar year minus 1900  
 3200 (*tm\_year*)) is related to a time represented as seconds since the Epoch, according to the  
 3201 expression below.

3202 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and  
 3203 the value is non-negative, the value is related to a Coordinated Universal Time name according  
 3204 to the C-language expression, where *tm\_sec*, *tm\_min*, *tm\_hour*, *tm\_yday*, and *tm\_year* are all  
 3205 integer types:

```
3206      tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 +
3207      (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 -
3208      (((tm_year-1)/100)*86400 + (((tm_year+299)/400)*86400
```

3209 The relationship between the actual time of day and the current value for seconds since the  
 3210 Epoch is unspecified.

3211 How any changes to the value of seconds since the Epoch are made to align to a desired  
 3212 relationship with the current actual time is implementation-defined. As represented in seconds 2  
 3213 since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

3214 **Note:** The last three terms of the expression add in a day for each year that follows a leap year  
 3215 starting with the first leap year since the Epoch. The first term adds a day every 4 years  
 3216 starting in 1973, the second subtracts a day back out every 100 years starting in 2001, and the  
 3217 third adds a day back in every 400 years starting in 2001. The divisions in the formula are  
 3218 integer divisions; that is, the remainder is discarded leaving only the integer quotient.

## 3219 4.15 Semaphore

3220 A minimum synchronization primitive to serve as a basis for more complex synchronization  
 3221 mechanisms to be defined by the application program.

3222 For the semaphores associated with the Semaphores option, a semaphore is represented as a  
 3223 shareable resource that has a non-negative integer value. When the value is zero, there is a  
 3224 (possibly empty) set of threads awaiting the availability of the semaphore.

3225 For the semaphores associated with the X/Open System Interface Extension (XSI), a semaphore  
 3226 is a positive integer (0 through 32767). The *semget()* function can be called to create a set or array  
 3227 of semaphores. A semaphore set can contain one or more semaphores up to an implementation-  
 3228 defined value.

### 3229 Semaphore Lock Operation

3230 An operation that is applied to a semaphore. If, prior to the operation, the value of the  
 3231 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and  
 3232 added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

### 3233 Semaphore Unlock Operation

3234 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in  
 3235 the set of threads awaiting the semaphore, then some thread from that set shall be removed from  
 3236 the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

## 3237 4.16 Thread-Safety

3238 Refer to the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.9, Threads.

## 3239 4.17 Tracing

3240 The trace system allows a traced process to have a selection of events created for it. Traces  
 3241 consist of streams of trace event types.

3242 A trace event type is identified on the one hand by a trace event type name, also referenced as a  
 3243 trace event name, and on the other hand by a trace event type identifier. A trace event name is a  
 3244 human-readable string. A trace event type identifier is an opaque identifier used by the trace  
 3245 system. There shall be a one-to-one relationship between trace event type identifiers and trace  
 3246 event names for a given trace stream and also for a given traced process. The trace event type  
 3247 identifier shall be generated automatically from a trace event name by the trace system either  
 3248 when a trace controller process invokes *posix\_trace\_trid\_eventid\_open()* or when an instrumented  
 3249 application process invokes *posix\_trace\_eventid\_open()*. Trace event type identifiers are used to  
 3250 filter trace event types, to allow interpretation of user data, and to identify the kind of trace point  
 3251 that generated a trace event.

3252 Each trace event shall be of a particular trace event type, and associated with a trace event type  
 3253 identifier. The execution of a trace point shall generate a trace event if a trace stream has been  
 3254 created and started for the process that executed the trace point and if the corresponding trace  
 3255 event type identifier is not ignored by filtering.

3256 A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a  
 3257 trace log is associated with the trace stream, except that:

- 3258 • For a trace stream, if no resources are available for the event, the event is lost.
- 3259 • For a trace log, if no resources are available for the event, or a flush operation does not  
 3260 succeed, the event is lost.

3261 A trace event recorded in an active trace stream may be retrieved by an application having the  
 3262 appropriate privileges.

3263 A trace event recorded in a trace log may be retrieved by an application having the appropriate  
 3264 privileges after opening the trace log as a pre-recorded trace stream, with the function  
 3265 *posix\_trace\_open()*.

3266 When a trace event is reported it is possible to retrieve the following:

- 3267 • A trace event type identifier
- 3268 • A timestamp
- 3269 • The process ID of the traced process, if the trace event is process-dependent
- 3270 • Any optional trace event data including its length

- 3271 • If the Threads option is supported, the thread ID, if the trace event is process-dependent
  - 3272 • The program address at which the trace point was invoked
- 3273 Trace events may be mapped from trace event types to trace event names. One such mapping  
 3274 shall be associated with each trace stream. An active trace stream is associated with a traced  
 3275 process, and also with its children if the Trace Inherit option is supported and also the  
 3276 inheritance policy is set to `_POSIX_TRACE_INHERIT`. Therefore each traced process has a  
 3277 mapping of the trace event names to trace event type identifiers that have been defined for that  
 3278 process.
- 3279 Traces can be recorded into either trace streams or trace logs.
- 3280 The implementation and format of a trace stream are unspecified. A trace stream need not be  
 3281 and generally is not persistent. A trace stream may be either active or pre-recorded:
- 3282 • An active trace stream is a trace stream that has been created and has not yet been shut  
 3283 down. It can be of one of the two following classes:
    - 3284 1. An active trace stream without a trace log that was created with the *posix\_trace\_create()*  
 3285 function
    - 3286 2. If the Trace Log option is supported, an active trace stream with a trace log that was  
 3287 created with the *posix\_trace\_create\_withlog()* function
  - 3288 • A pre-recorded trace stream is a trace stream that was opened from a trace log object using  
 3289 the *posix\_trace\_open()* function.
- 3290 An active trace stream can loop. This behavior means that when the resources allocated by the  
 3291 trace system for the trace stream are exhausted, the trace system reuses the resources associated  
 3292 with the oldest recorded trace events to record new trace events.
- 3293 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This  
 3294 operation causes the trace system to write trace events from the trace stream to the associated  
 3295 trace log, following the defined policies or using an explicit function call. After this operation,  
 3296 the trace system may reuse the resources associated with the flushed trace events.
- 3297 An active trace stream with or without a trace log can be cleared. This operation shall cause all  
 3298 the resources associated with this trace stream to be reinitialized. The trace stream shall behave  
 3299 as if it was returning from its creation, except that the mapping of trace event type identifiers to  
 3300 trace event names shall not be cleared. If a trace log was associated with this trace stream, the  
 3301 trace log shall also be reinitialized.
- 3302 A trace log shall be recorded when the *posix\_trace\_shutdown()* operation is invoked or during  
 3303 tracing, depending on the tracing strategy which is defined by a log policy. After the trace  
 3304 stream has been shut down, the trace information can be retrieved from the associated trace log  
 3305 using the same interface used to retrieve information from an active trace stream.
- 3306 For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance  
 3307 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together  
 3308 with all of its future children. The *posix\_pid* member of each trace event in a trace stream shall be  
 3309 the process ID of the traced process.
- 3310 Each trace point may be an implementation-defined action such as a context switch, or an  
 3311 application-programmed action such as a call to a specific operating system service (for  
 3312 example, *fork()*) or a call to *posix\_trace\_event()*.
- 3313 Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace  
 3314 events. By default, no trace events are filtered.

3315 The results of the tracing operations can be analyzed and monitored by a trace controller process  
 3316 or a trace analyzer process.

3317 Only the trace controller process has control of the trace stream it has created. The control of the  
 3318 operation of a trace stream is done using its corresponding trace stream identifier. The trace  
 3319 controller process is able to:

- 3320 • Initialize the attributes of a trace stream
- 3321 • Create the trace stream
- 3322 • Start and stop tracing
- 3323 • Know the mapping of the traced process
- 3324 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3325 • Shut the trace stream down

3326 A traced process may also be a trace controller process. Only the trace controller process can  
 3327 control its trace stream(s). A trace stream created by a trace controller process shall be shut  
 3328 down if its controller process terminates or executes another file.

3329 A trace controller process may also be a trace analyzer process. Trace analysis can be done  
 3330 concurrently with the traced process or can be done off-line, in the same or in a different  
 3331 platform.

## 3332 **4.18 Treatment of Error Conditions for Mathematical Functions**

3333 For all the functions in the `<math.h>` header, an application wishing to check for error situations  
 3334 should set `errno` to 0 and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On  
 3335 return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |`  
 3336 `FE_UNDERFLOW)` is non-zero, an error has occurred.

3337 The following error conditions are defined for all functions in the `<math.h>` header.

### 3338 **4.18.1 Domain Error**

3339 A “domain error” shall occur if an input argument is outside the domain over which the  
 3340 mathematical function is defined. The description of each function lists any required domain  
 3341 errors; an implementation may define additional domain errors, provided that such errors are  
 3342 consistent with the mathematical definition of the function.

3343 On a domain error, the function shall return an implementation-defined value; if the integer  
 3344 expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to `[EDOM]`; if  
 3345 the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “invalid”  
 3346 floating-point exception shall be raised.

3347 **4.18.2 Pole Error**

3348 A “pole error” occurs if the mathematical result of the function is an exact infinity (for example,  
3349  $\log(0.0)$ ).

3350 On a pole error, the function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or  
3351 `HUGE_VALL` according to the return type, with the same sign as the correct value of the  
3352 function; if the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero, *errno* shall  
3353 be set to `[ERANGE]`; if the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is  
3354 non-zero, the “divide-by-zero” floating-point exception shall be raised.

3355 **4.18.3 Range Error**

3356 A “range error” shall occur if the finite mathematical result of the function cannot be  
3357 represented in an object of the specified type, due to extreme magnitude.

3358 **4.18.3.1 Result Overflows**

3359 A floating result overflows if the magnitude of the mathematical result is finite but so large that  
3360 the mathematical result cannot be represented without extraordinary roundoff error in an object  
3361 of the specified type. If a floating result overflows and default rounding is in effect, then the  
3362 function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or `HUGE_VALL`  
3363 according to the return type, with the same sign as the correct value of the function; if the integer  
3364 expression (`math_errhandling` & `MATH_ERRNO`) is non-zero, *errno* shall be set to `[ERANGE]`; if  
3365 the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the “overflow”  
3366 floating-point exception shall be raised.

3367 **4.18.3.2 Result Underflows**

3368 The result underflows if the magnitude of the mathematical result is so small that the  
3369 mathematical result cannot be represented, without extraordinary roundoff error, in an object of  
3370 the specified type. If the result underflows, the function shall return an implementation-defined  
3371 value whose magnitude is no greater than the smallest normalized positive number in the  
3372 specified type; if the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
3373 whether *errno* is set to `[ERANGE]` is implementation-defined; if the integer expression  
3374 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, whether the “underflow” floating-point  
3375 exception is raised is implementation-defined.

3376 **4.19 Treatment of NaN Arguments for the Mathematical Functions**

3377 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,  
3378 except where stated otherwise.

3379 If a function with one or more NaN arguments returns a NaN result, the result should be the  
3380 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3381 On implementations that support the IEC 60559:1989 standard floating point, functions with  
3382 signaling NaN argument(s) shall be treated as if the function were called with an argument that  
3383 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3384 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are  
3385 evaluated during the function call.

3386 On implementations that support the IEC 60559:1989 standard floating point, for those  
3387 functions that do not have a documented domain error, the following shall apply:



3388           These functions shall fail if:

3389           Domain Error     Any argument is a signaling NaN.

3390           Either, the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero and *errno*

3391           shall be set to [EDOM], or the integer expression (`math_errhandling` & `MATH_ERREXCEPT`)

3392           is non-zero and the invalid floating-point exception shall be raised.

## 3393 4.20 Utility

3394           A utility program shall be either an executable file, such as might be produced by a compiler or

3395           linker system from computer source code, or a file of shell source code, directly interpreted by

3396           the shell. The program may have been produced by the user, provided by the system

3397           implementor, or acquired from an independent distributor.

3398           The system may implement certain utilities as shell functions (see the Shell and Utilities volume

3399           of IEEE Std 1003.1-2001, Section 2.9.5, Function Definition Command) or built-in utilities, but

3400           only an application that is aware of the command search order described in the Shell and

3401           Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution or of

3402           performance characteristics can discern differences between the behavior of such a function or

3403           built-in utility and that of an executable file.

## 3404 4.21 Variable Assignment

3405           In the shell command language, a word consisting of the following parts:

3406           `varname=value`

3407           When used in a context where assignment is defined to occur and at no other time, the *value*

3408           (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

3409           **Note:**     For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section

3410                       2.9.1, Simple Commands.

3411           The *varname* and *value* parts shall meet the requirements for a name and a word, respectively,

3412           except that they are delimited by the embedded unquoted equals-sign, in addition to other

3413           delimiters.

3414           **Note:**     Additional delimiters are described in the Shell and Utilities volume of IEEE Std 1003.1-2001,

3415                       Section 2.3, Token Recognition.

3416           When a variable assignment is done, the variable shall be created if it did not already exist. If

3417           *value* is not specified, the variable shall be given a null value.

3418           **Note:**     An alternative form of variable assignment:

3419           `symbol=value`

3420           (where *symbol* is a valid word delimited by an equals-sign, but not a valid name) produces

3421           unspecified results. The form *symbol=value* is used by the KornShell *name[expression]=value*

3422           syntax.



## File Format Notation

3424

3425 The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility  
 3426 descriptions use a syntax to describe the data organization within the files, when that  
 3427 organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces  
 3428 volume of IEEE Std 1003.1-2001 *printf()* function, as described in this chapter. When used in  
 3429 STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that  
 3430 could have been used to write the text to be read, not a format that could be used by the System  
 3431 Interfaces volume of IEEE Std 1003.1-2001 *scanf()* function to read the input file.

3432 The description of an individual record is as follows:

3433       "*<format>*", [*<arg1>*, *<arg2>*, . . . , *<argn>*]

3434 The *format* is a character string that contains three types of objects defined below:

- 3435       1. *Characters* that are not "escape sequences" or "conversion specifications", as described  
 3436       below, shall be copied to the output.
- 3437       2. *Escape Sequences* represent non-graphic characters.
- 3438       3. *Conversion Specifications* specify the output format of each argument; see below.

3439 The following characters have the following special meaning in the format string:

3440       ' ' (An empty character position.) Represents one or more *<blank>*s.

3441       Δ Represents exactly one *<space>*.

3442 Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

Table 5-1 Escape Sequences and Associated Actions

Escape Sequence	Represents Character	Terminal Action
'\\'	backslash	Print the character '\\ '.
'\a'	alert	Attempt to alert the user through audible or visible notification.
'\b'	backspace	Move the printing position to one column before the current position, unless the current position is the start of a line.
'\f'	form-feed	Move the printing position to the initial printing position of the next logical page.
'\n'	newline	Move the printing position to the start of the next line.
'\r'	carriage-return	Move the printing position to the start of the current line.
'\t'	tab	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
'\v'	vertical-tab	Move the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

Each conversion specification is introduced by the percent-sign character ('%'). After the character '%', the following shall appear in sequence:

**flags** Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

**field width** An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag ('-'), described below, has been given) to the field width.

**precision** Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversion specifiers (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversion specifiers, the maximum number of significant digits for the *g* conversion specifier; or the maximum number of bytes to be written from a string in the *s* conversion specifier. The precision shall take the form of a period ('.') followed by a decimal digit string; a null digit string is treated as zero.

**conversion specifier characters**

A conversion specifier character (see below) that indicates the type of conversion to be applied.

The *flag* characters and their meanings are:

– The result of the conversion shall be left-justified within the field.

– The result of a signed conversion shall always begin with a sign ('+' or '-').

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

# The value shall be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s* conversion specifiers, the behavior is undefined. For the *o* conversion specifier, it shall increase the precision to force the first digit of the result to be a zero. For *x* or *X* conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively. For

3488		e, E, f, g, and G conversion specifiers, the result shall always contain a radix
3489		character, even if no digits follow the radix character. For g and G conversion
3490		specifiers, trailing zeros shall not be removed from the result as they usually are.
3491	0	For d, i, o, u, x, X, e, E, f, g, and G conversion specifiers, leading zeros (following
3492		any indication of sign or base) shall be used to pad to the field width; no space
3493		padding is performed. If the '0' and '-' flags both appear, the '0' flag shall be
3494		ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the
3495		'0' flag shall be ignored. For other conversion specifiers, the behavior is
3496		undefined.
3497		Each conversion specifier character shall result in fetching zero or more arguments. The results
3498		are undefined if there are insufficient arguments for the format. If the format is exhausted while
3499		arguments remain, the excess arguments shall be ignored.
3500		The conversion specifiers and their meanings are:
3501	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i), unsigned octal
3502		(o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and
3503		i specifiers shall convert to signed decimal in the style "[−]dddd". The x
3504		conversion specifier shall use the numbers and letters "0123456789abcdef" and
3505		the X conversion specifier shall use the numbers and letters
3506		"0123456789ABCDEF". The <i>precision</i> component of the argument shall specify
3507		the minimum number of digits to appear. If the value being converted can be
3508		represented in fewer digits than the specified minimum, it shall be expanded with
3509		leading zeros. The default precision shall be 1. The result of converting a zero
3510		value with a precision of 0 shall be no characters. If both the field width and
3511		precision are omitted, the implementation may precede, follow, or precede and
3512		follow numeric arguments of types d, i, and u with <blank>s; arguments of type o
3513		(octal) may be preceded with leading zeros.
3514	f	The floating-point number argument shall be written in decimal notation in the
3515		style [−]ddd.ddd, where the number of digits after the radix character (shown here
3516		as a decimal point) shall be equal to the <i>precision</i> specification. The <i>LC_NUMERIC</i>
3517		locale category shall determine the radix character to use in this format. If the
3518		<i>precision</i> is omitted from the argument, six digits shall be written after the radix
3519		character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3520	e,E	The floating-point number argument shall be written in the style [−]d.ddde±dd (the
3521		symbol '±' indicates either a plus or minus sign), where there is one digit before
3522		the radix character (shown here as a decimal point) and the number of digits after
3523		it is equal to the precision. The <i>LC_NUMERIC</i> locale category shall determine the
3524		radix character to use in this format. When the precision is missing, six digits shall
3525		be written after the radix character; if the precision is 0, no radix character shall
3526		appear. The E conversion specifier shall produce a number with E instead of e
3527		introducing the exponent. The exponent shall always contain at least two digits.
3528		However, if the value to be written requires an exponent greater than two digits,
3529		additional exponent digits shall be written as necessary.
3530	g,G	The floating-point number argument shall be written in style f or e (or in style F or
3531		E in the case of a G conversion specifier), with the precision specifying the number
3532		of significant digits. The style used depends on the value converted: style e (or E)
3533		shall be used only if the exponent resulting from the conversion is less than −4 or
3534		greater than or equal to the precision. Trailing zeros are removed from the result. A
3535		radix character shall appear only if it is followed by a digit.

3536	c	The integer argument shall be converted to an <b>unsigned char</b> and the resulting
3537		byte shall be written.
3538	s	The argument shall be taken to be a string and bytes from the string shall be
3539		written until the end of the string or the number of bytes indicated by the <i>precision</i>
3540		specification of the argument is reached. If the precision is omitted from the
3541		argument, it shall be taken to be infinite, so all bytes up to the end of the string
3542		shall be written.
3543	%	Write a ' % ' character; no argument is converted.
3544		In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of
3545		a conversion is wider than the field width, the field is simply expanded to contain the conversion
3546		result. The term “field width” should not be confused with the term “precision” used in the
3547		description of %s.

### 3548 Examples

3549 To represent the output of a program that prints a date and time in the form Sunday, July 3,  
3550 10:02, where *weekday* and *month* are strings:

3551 " %s, %s %d, %d: %.2d \n" <weekday>, <month>, <day>, <hour>, <min>

3552 To show '  $\pi$  ' written to 5 decimal places:

3553 "pi = %.5f \n", <value of  $\pi$ >

3554 To show an input file format consisting of five colon-separated fields:

3555 " %s: %s: %s: %s: %s \n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>

# Character Set

## 6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of IEEE Std 1003.1-2001. The first eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the characters are defined in the ISO/IEC 10646-1:2000 standard.

**Table 6-1** Portable Character Set

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	–	<U002D>	HYPHEN-MINUS
<hyphen>	–	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE

3597

3598

3599

3600

3601

3602

3603

3604

3605

3606

3607

3608

3609

3610

3611

3612

3613

3614

3615

3616

3617

3618

3619

3620

3621

3622

3623

3624

3625

3626

3627

3628

3629

3630

3631

3632

3633

3634

3635

3636

3637

3638

3639

3640

3641

3642

3643

3644

3645

Symbolic Name	Glyph	UCS	Description
<four>	4	<U0034>	DIGIT FOUR
<five>	5	<U0035>	DIGIT FIVE
<six>	6	<U0036>	DIGIT SIX
<seven>	7	<U0037>	DIGIT SEVEN
<eight>	8	<U0038>	DIGIT EIGHT
<nine>	9	<U0039>	DIGIT NINE
<colon>	:	<U003A>	COLON
<semicolon>	;	<U003B>	SEMICOLON
<less-than-sign>	<	<U003C>	LESS-THAN SIGN
<equals-sign>	=	<U003D>	EQUALS SIGN
<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
<question-mark>	?	<U003F>	QUESTION MARK
<commercial-at>	@	<U0040>	COMMERCIAL AT
<A>	A	<U0041>	LATIN CAPITAL LETTER A
<B>	B	<U0042>	LATIN CAPITAL LETTER B
<C>	C	<U0043>	LATIN CAPITAL LETTER C
<D>	D	<U0044>	LATIN CAPITAL LETTER D
<E>	E	<U0045>	LATIN CAPITAL LETTER E
<F>	F	<U0046>	LATIN CAPITAL LETTER F
<G>	G	<U0047>	LATIN CAPITAL LETTER G
<H>	H	<U0048>	LATIN CAPITAL LETTER H
<I>	I	<U0049>	LATIN CAPITAL LETTER I
<J>	J	<U004A>	LATIN CAPITAL LETTER J
<K>	K	<U004B>	LATIN CAPITAL LETTER K
<L>	L	<U004C>	LATIN CAPITAL LETTER L
<M>	M	<U004D>	LATIN CAPITAL LETTER M
<N>	N	<U004E>	LATIN CAPITAL LETTER N
<O>	O	<U004F>	LATIN CAPITAL LETTER O
<P>	P	<U0050>	LATIN CAPITAL LETTER P
<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
<R>	R	<U0052>	LATIN CAPITAL LETTER R
<S>	S	<U0053>	LATIN CAPITAL LETTER S
<T>	T	<U0054>	LATIN CAPITAL LETTER T
<U>	U	<U0055>	LATIN CAPITAL LETTER U
<V>	V	<U0056>	LATIN CAPITAL LETTER V
<W>	W	<U0057>	LATIN CAPITAL LETTER W
<X>	X	<U0058>	LATIN CAPITAL LETTER X
<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
<backslash>	\	<U005C>	REVERSE SOLIDUS
<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
<low-line>	_	<U005F>	LOW LINE
<underscore>	_	<U005F>	LOW LINE



3646

3647

3648

3649

3650

3651

3652

3653

3654

3655

3656

3657

3658

3659

3660

3661

3662

3663

3664

3665

3666

3667

3668

3669

3670

3671

3672

3673

3674

3675

3676

3677

3678

3679

3680

Symbolic Name	Glyph	UCS	Description
<grave-accent>	`	<U0060>	GRAVE ACCENT
<a>	a	<U0061>	LATIN SMALL LETTER A
<b>	b	<U0062>	LATIN SMALL LETTER B
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>	{	<U007B>	LEFT CURLY BRACKET
<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>		<U007C>	VERTICAL LINE
<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

3681

3682

3683

IEEE Std 1003.1-2001 uses character names other than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of IEEE Std 1003.1-2001.

3684

3685

3686

3687

3688

Table 6-1 (on page 113) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name. Characters defined in Table 6-2 (on page 118) may also be used in character set description files.

3689

3690

IEEE Std 1003.1-2001 places only the following requirements on the encoded values of the characters in the portable character set:

3691

3692

3693

3694

3695

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, if an application accesses any pair of locales where the character encodings differ, or accesses data from an application running in a locale which has different encodings from the application's current locale, the results are unspecified.

- 3696 • The encoded values associated with the digits 0 to 9 shall be such that the value of each
- 3697 character after 0 shall be one greater than the value of the previous character.
- 3698 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3699 • The encoded values associated with the members of the portable character set are each
- 3700 represented in a single byte. Moreover, if the value is stored in an object of C-language type
- 3701 **char**, it is guaranteed to be positive (except the NUL, which is always zero).
- 3702 Conforming implementations shall support certain character and character set attributes, as
- 3703 defined in Section 7.2 (on page 124).

## 3704 6.2 Character Encoding

3705 The POSIX locale contains the characters in Table 6-1 (on page 113), which have the properties  
 3706 listed in Section 7.3.1 (on page 126). In other locales, the presence, meaning, and representation  
 3707 of any additional characters are locale-specific.

3708 In locales other than the POSIX locale, a character may have a state-dependent encoding. There  
 3709 are two types of these encodings:

- 3710 • A single-shift encoding (where each character not in the initial shift state is preceded by a
- 3711 shift code) can be defined if each shift-code and character sequence is considered a multi-
- 3712 byte character. This is done using the concatenated-constant format in a character set
- 3713 description file, as described in Section 6.4 (on page 117). If the implementation supports a
- 3714 character encoding of this type, all of the standard utilities in the Shell and Utilities volume of
- 3715 IEEE Std 1003.1-2001 shall support it. Use of a single-shift encoding with any of the functions
- 3716 in the System Interfaces volume of IEEE Std 1003.1-2001 that do not specifically mention the
- 3717 effects of state-dependent encoding is implementation-defined.
- 3718 • A locking-shift encoding (where the state of the character is determined by a shift code that
- 3719 may affect more than the single character following it) cannot be defined with the current
- 3720 character set description file format. Use of a locking-shift encoding with any of the standard
- 3721 utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 or with any of the functions
- 3722 in the System Interfaces volume of IEEE Std 1003.1-2001 that do not specifically mention the
- 3723 effects of state-dependent encoding is implementation-defined.

3724 While in the initial shift state, all characters in the portable character set shall retain their usual  
 3725 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the  
 3726 sequence shall be a function of the current shift state. A byte with all bits zero shall be  
 3727 interpreted as the null character independent of shift state. Thus a byte with all bits zero shall  
 3728 never occur in the second or subsequent bytes of a character.

3729 The maximum allowable number of bytes in a character in the current locale shall be indicated  
 3730 by {MB\_CUR\_MAX}, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a  
 3731 character set description file; see Section 6.4 (on page 117). The implementation's maximum  
 3732 number of bytes in a character shall be defined by the C-language macro {MB\_LEN\_MAX}.

### 3733 6.3 C Language Wide-Character Codes

3734 In the shell, the standard utilities are written so that the encodings of characters are described by  
 3735 the locale's *LC\_CTYPE* definition (see Section 7.3.1 (on page 126)) and there is no differentiation  
 3736 between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C  
 3737 language, a differentiation is made. To ease the handling of variable length characters, the C  
 3738 language has introduced the concept of wide-character codes.

3739 All wide-character codes in a given process consist of an equal number of bits. This is in contrast  
 3740 to characters, which can consist of a variable number of bytes. The byte or byte sequence that  
 3741 represents a character can also be represented as a wide-character code. Wide-character codes  
 3742 thus provide a uniform size for manipulating text data. A wide-character code having all bits  
 3743 zero is the null wide-character code (see Section 3.246 (on page 69)), and terminates wide-  
 3744 character strings (see Section 3.432 (on page 94)). The wide-character value for each member of  
 3745 the portable character set shall equal its value when used as the lone character in an integer  
 3746 character constant. Wide-character codes for other characters are locale and implementation-  
 3747 defined. State shift bytes shall not have a wide-character code representation. This standard  
 3748 provides no means of defining a wide-character codeset.

2  
2

### 3749 6.4 Character Set Description File

3750 Implementations shall provide a character set description file for at least one coded character set  
 3751 supported by the implementation. These files are referred to elsewhere in IEEE Std 1003.1-2001  
 3752 as *charmap* files. It is implementation-defined whether or not users or applications can provide  
 3753 additional character set description files.

3754 IEEE Std 1003.1-2001 does not require that multiple character sets or codesets be supported.  
 3755 Although multiple charmap files are supported, it is the responsibility of the implementation to  
 3756 provide the file or files; if only one is provided, only that one is accessible using the *localedef*  
 3757 utility's *-f* option.

3758 Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard  
 3759 position values as the encoding values, shall define characteristics for the coded character set  
 3760 and the encoding for the characters specified in Table 6-1 (on page 113), and may define  
 3761 encoding for additional characters supported by the implementation. Other information about  
 3762 the coded character set may also be in the file. Coded character set character values shall be  
 3763 defined using symbolic character names followed by character encoding values.

3764 Each symbolic name specified in Table 6-1 (on page 113) shall be included in the file and shall be  
 3765 mapped to a unique coding value, except as noted below. The glyphs ' { , ' } ', ' \_ ', ' - ', ' / ',  
 3766 ' \ ', ' . ', and ' ^ ' have more than one symbolic name; all symbolic names for each such glyph  
 3767 shall be included, each with identical encoding. If some or all of the control characters identified  
 3768 in Table 6-2 (on page 118) are supported by the implementation, the symbolic names and their  
 3769 corresponding encoding values shall be included in the file. Some of the encodings associated  
 3770 with the symbolic names in Table 6-2 (on page 118) may be the same as characters found in Table  
 3771 6-1 (on page 113); both names shall be provided for each encoding.

Table 6-2 Control Character Set

<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>
<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
<DC1>	<EM>	<FF>	<IS3>	<SO>	<VT>

The following declarations can precede the character definitions. Each shall consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one or more <blank>s, followed by the value to be assigned to the symbol.

**<code\_set\_name>** The name of the coded character set for which the character set description file is defined. The characters of the name shall be taken from the set of characters with visible glyphs defined in Table 6-1 (on page 113).

**<mb\_cur\_max>** The maximum number of bytes in a multi-byte character. This shall default to 1.

**<mb\_cur\_min>** An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set. On XSI-conformant systems, <mb\_cur\_min> shall always be 1.

**<escape\_char>** The character used to indicate that the characters following shall be interpreted in a special way, as defined later in this section. This shall default to backslash ('\''), which is the character used in all the following text and examples, unless otherwise noted.

**<comment\_char>** The character that, when placed in column 1 of a charmap line, is used to indicate that the line shall be ignored. The default character shall be the number sign ('#').

The character set mapping definitions shall be all the lines immediately following an identifier line containing the string "CHARMAP" starting in column 1, and preceding a trailer line containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a <comment\_char> in the first column shall be ignored. Each non-comment line of the character set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file) shall be in either of two forms:

```
"%s %s %s\n", <symbolic-name>, <encoding>, <comments>
```

or:

```
"%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,
<encoding>, <comments>
```

In the first format, the line in the character set mapping definition shall define a single symbolic name and a corresponding encoding. A symbolic name is one or more characters from the set shown with visible glyphs in Table 6-1 (on page 113), enclosed between angle brackets. A character following an escape character is interpreted as itself; for example, the sequence "<\\>>" represents the symbolic name "<\\>" enclosed between angle brackets.

In the second format, the line in the character set mapping definition shall define a range of one or more symbolic names. In this form, the symbolic names shall consist of zero or more non-numeric characters from the set shown with visible glyphs in Table 6-1 (on page 113), followed by an integer formed by one or more decimal digits. Both integers shall contain the same number of digits. The characters preceding the integer shall be identical in the two symbolic names, and

the integer formed by the digits in the second symbolic name shall be equal to or greater than the integer formed by the digits in the first name. This shall be interpreted as a series of symbolic names formed from the common part and each of the integers between the first and the second integer, inclusive. As an example, `<j0101>...<j0104>` is interpreted as the symbolic names `<j0101>`, `<j0102>`, `<j0103>`, and `<j0104>`, in that order.

A character set mapping definition line shall exist for all symbolic names specified in Table 6-1 (on page 113), and shall define the coded character value that corresponds to the character indicated in the table, or the coded character value that corresponds to the control character symbolic name. If the control characters commonly associated with the symbolic names in Table 6-2 (on page 118) are supported by the implementation, the symbolic name and the corresponding encoding value shall be included in the file. Additional unique symbolic names may be included. A coded character value can be represented by more than one symbolic name.

The encoding part is expressed as one (for single-byte character values) or more concatenated decimal, octal, or hexadecimal constants in the following formats:

```
%cd%u", <escape_char>, <decimal byte value>
%cx%x", <escape_char>, <hexadecimal byte value>
%co", <escape_char>, <octal byte value>
```

Decimal constants shall be represented by two or three decimal digits, preceded by the escape character and the lowercase letter 'd'; for example, `"\d05"`, `"\d97"`, or `"\d143"`. Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape character and the lowercase letter 'x'; for example, `"\x05"`, `"\x61"`, or `"\x8f"`. Octal constants shall be represented by two or three octal digits, preceded by the escape character; for example, `"\05"`, `"\141"`, or `"\217"`. In a portable charmap file, each constant represents an 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the same type, and interpreted in byte order from first to last with the least significant byte of the multi-byte character specified by the last constant. The manner in which these constants are represented in the character stored in the system is implementation-defined. (This notation was chosen for reasons of portability. There is no requirement that the internal representation in the computer memory be in this same order.) Omitting bytes from a multi-byte character definition produces undefined results.

In lines defining ranges of symbolic names, the encoded value shall be the value for the first symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names defined by the range shall have encoding values in increasing order. Bytes shall be treated as unsigned octets, and carry shall be propagated between the bytes as necessary to represent the range. However, because this causes a null byte in the second or subsequent bytes of a character, such a declaration should not be specified. For example, the line:

```
<j0101>...<j0104>  \d129\d254
```

is interpreted as:

```
<j0101>          \d129\d254
<j0102>          \d129\d255
<j0103>          \d130\d00
<j0104>          \d130\d01
```

The expanded declaration of the symbol `<j0103>` in the above example is an invalid specification, because it contains a null byte in the second byte of a character.

The comment is optional.

This standard provides no means of defining a wide-character codeset.

The following declarations can follow the character set mapping definitions (after the "END CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in column 1, followed by the value(s) to be associated to the keyword, as defined below.

**WIDTH** A non-negative integer value defining the column width (see Section 3.103 (on page 49)) for the printable characters in the coded character set specified in Table 6-1 (on page 113) and Table 6-2 (on page 118). Coded character set character values shall be defined using symbolic character names followed by column width values. Defining a character with more than one **WIDTH** produces undefined results. The **END WIDTH** keyword shall be used to terminate the **WIDTH** definitions. Specifying the width of a non-printable character in a **WIDTH** declaration produces undefined results.

**WIDTH\_DEFAULT** A non-negative integer value defining the default column width for any printable character not listed by one of the **WIDTH** keywords. If no **WIDTH\_DEFAULT** keyword is included in the charmap, the default character width shall be 1.

### Example

After the "END CHARMAP" statement, a syntax for a width definition would be:

```
WIDTH
<A> 1
<B> 1
<C>...<Z> 1
...
<fool>...<foon> 2
...
END WIDTH
```

In this example, the numerical code point values represented by the symbols <A> and <B> are assigned a width of 1. The code point values <C> to <Z> inclusive (<C>, <D>, <E>, and so on) are also assigned a width of 1. Using <A>...<Z> would have required fewer lines, but the alternative was shown to demonstrate flexibility. The keyword **WIDTH\_DEFAULT** could have been added as appropriate.

## 6.4.1 State-Dependent Character Encodings

This section addresses the use of state-dependent character encodings (that is, those in which the encoding of a character is dependent on one or more shift codes that may precede it).

A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined in the charmap format if each shift-code/character sequence is considered a multi-byte character, defined using the concatenated-constant format described in Section 6.4 (on page 117). If the implementation supports a character encoding of this type, all of the standard utilities shall support it. A locking-shift encoding (where the state of the character is determined by a shift code that may affect more than the single character following it) could be defined with an extension to the charmap format described in Section 6.4 (on page 117). If the implementation supports a character encoding of this type, any of the standard utilities that describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

1. The utility shall process the statefully encoded data as a concatenation of state-independent characters. The presence of redundant locking shifts shall not affect the comparison of two statefully encoded strings.

3909           2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall  
3910           produce output that contains locking shifts at the beginning or end of the resulting data, if  
3911           appropriate, to retain correct state information.





3913

**3914 7.1 General**

3915 A locale is the definition of the subset of a user's environment that depends on language and  
 3916 cultural conventions. It is made up from one or more categories. Each category is identified by  
 3917 its name and controls specific aspects of the behavior of components of the system. Category  
 3918 names correspond to the following environment variable names:

3919 *LC\_CTYPE* Character classification and case conversion.

3920 *LC\_COLLATE* Collation order.

3921 *LC\_MONETARY* Monetary formatting.

3922 *LC\_NUMERIC* Numeric, non-monetary formatting.

3923 *LC\_TIME* Date and time formats.

3924 *LC\_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

3925 The standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 shall base their  
 3926 behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each  
 3927 utility. The behavior of some of the C-language functions defined in the System Interfaces  
 3928 volume of IEEE Std 1003.1-2001 shall also be modified based on the current locale, as defined by  
 3929 the last call to *setlocale()*.

3930 Locales other than those supplied by the implementation can be created via the *localedef* utility,  
 3931 provided that the *\_POSIX2\_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not  
 3932 provided, all implementations conforming to the System Interfaces volume of  
 3933 IEEE Std 1003.1-2001 shall provide one or more locales that behave as described in this chapter.  
 3934 The input to the utility is described in Section 7.3 (on page 124). The value that is used to specify  
 3935 a locale when using environment variables shall be the string specified as the *name* operand to  
 3936 the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as  
 3937 identifiers for the POSIX locale (see Section 7.2 (on page 124)). When the value of a locale  
 3938 environment variable begins with a slash ('/'), it shall be interpreted as the pathname of the  
 3939 locale definition; the type of file (regular, directory, and so on) used to store the locale definition  
 3940 is implementation-defined. If the value does not begin with a slash, the mechanism used to  
 3941 locate the locale is implementation-defined.

3942 If different character sets are used by the locale categories, the results achieved by an application  
 3943 utilizing these categories are undefined. Likewise, if different codesets are used for the data  
 3944 being processed by interfaces whose behavior is dependent on the current locale, or the codeset  
 3945 is different from the codeset assumed when the locale was created, the result is also undefined.

3946 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)  
 3947 with the appropriate value. If the function is invoked with an empty string, such as:

3948 `setlocale(LC_ALL, "");`

3949 the value of the corresponding environment variable is used. If the environment variable is  
 3950 unset or is set to the empty string, the implementation shall set the appropriate environment as  
 3951 defined in Chapter 8 (on page 161).

## 7.2 POSIX Locale

Conforming systems shall provide a POSIX locale, also known as the C locale. The behavior of standard utilities and functions in the POSIX locale shall be as if the locale was defined via the *localedef* utility with input data from the POSIX locale tables in Section 7.3.

The tables in Section 7.3 describe the characteristics and behavior of the POSIX locale for data consisting entirely of characters from the portable character set and the control character set. For other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall be the default locale when the *setlocale()* function is not called.

The POSIX locale can be specified by assigning to the appropriate environment variables the values "C" or "POSIX".

All implementations shall define a locale as the default locale, to be invoked when no environment variables are set, or set to the empty string. This default locale can be the POSIX locale or any other implementation-defined locale. Some implementations may provide facilities for local installation administrators to set the default locale, customizing it for each location. IEEE Std 1003.1-2001 does not require such a facility.

## 7.3 Locale Definition

The capability to specify additional locales to those provided by an implementation is optional, denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only implementation-supplied locales are available. Such locales shall be documented using the format specified in this section.

Locales can be described with the file format presented in this section. The file format is that accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the "locale definition file", but no locales shall be affected by this file unless it is processed by *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility shall apply to *localedef* or to any other similar utility used to install locale information using the locale definition file format described here.

The locale definition file shall contain one or more locale category source definitions, and shall not contain more than one definition for the same locale category. If the file contains source definitions for more than one category, implementation-defined categories, if present, shall appear after the categories defined by Section 7.1 (on page 123). A category source definition contains either the definition of a category or a **copy** directive. For a description of the **copy** directive, see *localedef*. In the event that some of the information for a locale category, as specified in this volume of IEEE Std 1003.1-2001, is missing from the locale source definition, the behavior of that category, if it is referenced, is unspecified.

A category source definition shall consist of a category header, a category body, and a category trailer. A category header shall consist of the character string naming of the category, beginning with the characters `LC_`. The category trailer shall consist of the string "END", followed by one or more <blank>s and the string used in the corresponding category header.

The category body shall consist of one or more lines of text. Each line shall contain an identifier, optionally followed by one or more operands. Identifiers shall be either keywords, identifying a particular locale element, or collating elements. In addition to the keywords defined in this volume of IEEE Std 1003.1-2001, the source can contain implementation-defined keywords. Each keyword within a locale shall have a unique name (that is, two categories cannot have a commonly-named keyword); no keyword shall start with the characters `LC_`. Identifiers shall be separated from the operands by one or more <blank>s.

Operands shall be characters, collating elements, or strings of characters. Strings shall be enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the *<escape character>*, described below. When a keyword is followed by more than one operand, the operands shall be separated by semicolons; *<blank>*s shall be allowed both before and after a semicolon.

The first category header in the file can be preceded by a line modifying the comment character. It shall have the following format, starting in column 1:

```
"comment_char %c\n", <comment character>
```

The comment character shall default to the number sign ('#'). Blank lines and lines containing the *<comment character>* in the first position shall be ignored.

The first category header in the file can be preceded by a line modifying the escape character to be used in the file. It shall have the following format, starting in column 1:

```
"escape_char %c\n", <escape character>
```

The escape character shall default to backslash, which is the character used in all examples shown in this volume of IEEE Std 1003.1-2001.

A line can be continued by placing an escape character as the last character on the line; this continuation character shall be discarded from the input. Although the implementation need not accept any one portion of a continued line with a length exceeding {LINE\_MAX} bytes, it shall place no limits on the accumulated length of the continued line. Comment lines shall not be continued on a subsequent line using an escaped *<newline>*.

Individual characters, characters in strings, and collating elements shall be represented using symbolic names, as defined below. In addition, characters can be represented using the characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic notation is used, the resultant locale definitions are in many cases not portable between systems. The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when used to represent itself it shall be preceded by the escape character. The following rules apply to character representation:

1. A character can be represented via a symbolic name, enclosed within angle brackets '<' and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic name defined in the charmap file specified via the *localedef -f* option, and it shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the charmap file shall constitute an error, unless the category is *LC\_CTYPE* or *LC\_COLLATE*, in which case it shall constitute a warning condition (see *localedef* for a description of actions resulting from errors and warnings). The specification of a symbolic name in a **collating-element** or **collating-symbol** section that duplicates a symbolic name in the charmap file (if present) shall be an error. Use of the escape character or a right angle bracket within a symbolic name is invalid unless the character is preceded by the escape character.

For example:

```
<c>;<c-cedilla> " <M><a><y>"
```

2. A character in the portable character set can be represented by the character itself, in which case the value of the character is implementation-defined. (Implementations may allow other characters to be represented as themselves, but such locale definitions are not portable.) Within a string, the double-quote character, the escape character, and the right angle bracket character shall be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters:

4043                   ,       ;       <       >       escape\_char

4044                   shall be escaped to be interpreted as the character itself.

4045                   For example:

4046                   c       "May"

4047                   3. A character can be represented as an octal constant. An octal constant shall be specified as  
4048                   the escape character followed by two or three octal digits. Each constant shall represent a  
4049                   byte value. Multi-byte values can be represented by concatenated constants specified in  
4050                   byte order with the last constant specifying the least significant byte of the character.

4051                   For example:

4052                   \143;\347;\143\150       "\115\141\171"

4053                   4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be  
4054                   specified as the escape character followed by an 'x' followed by two hexadecimal digits.  
4055                   Each constant shall represent a byte value. Multi-byte values can be represented by  
4056                   concatenated constants specified in byte order with the last constant specifying the least  
4057                   significant byte of the character.

4058                   For example:

4059                   \x63;\xe7;\x63\x68       "\x4d\x61\x79"

4060                   5. A character can be represented as a decimal constant. A decimal constant shall be specified  
4061                   as the escape character followed by a 'd' followed by two or three decimal digits. Each  
4062                   constant represents a byte value. Multi-byte values can be represented by concatenated  
4063                   constants specified in byte order with the last constant specifying the least significant byte  
4064                   of the character.

4065                   For example:

4066                   \d99;\d231;\d99\d104       "\d77\d97\d121"

4067                   Implementations may accept single-digit octal, decimal, or hexadecimal constants following the  
4068                   escape character. Only characters existing in the character set for which the locale definition is  
4069                   created shall be specified, whether using symbolic names, the characters themselves, or octal,  
4070                   decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the  
4071                   charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not  
4072                   present in the charmap file can be specified and shall be ignored, as specified under item 1  
4073                   above.

### 4074   7.3.1   LC\_CTYPE

4075                   The *LC\_CTYPE* category shall define character classification, case conversion, and other  
4076                   character attributes. In addition, a series of characters can be represented by three adjacent  
4077                   periods representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted as  
4078                   meaning that all values between the values preceding and following it represent valid  
4079                   characters. The ellipsis specification shall be valid only within a single encoded character set;  
4080                   that is, within a group of characters of the same size. An ellipsis shall be interpreted as including  
4081                   in the list all characters with an encoded value higher than the encoded value of the character  
4082                   preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

4083                   For example:

4084                   \x30;...;\x39;

includes in the character class all characters with encoded values between the endpoints.

The following keywords shall be recognized. In the descriptions, the term “automatically included” means that it shall not be an error either to include or omit any of the referenced characters; the implementation provides them if missing (even if the entire keyword is missing) and accepts them silently if present. When the implementation automatically includes a missing character, it shall have an encoded value dependent on the charmap file in effect (see the description of the *localedef* -f option); otherwise, it shall have a value derived from an implementation-defined character mapping.

The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included characters. These only need to be specified if the character values (that is, encoding) differ from the implementation default values. It is not possible to define a locale without these automatically included characters unless some implementation extension is used to prevent their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and, thus, it might not be possible for conforming applications to work properly.

**copy** Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

**upper** Define characters to be classified as uppercase letters.

In the POSIX locale, the 26 uppercase letters shall be included:

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

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as defined in Section 6.4 (on page 117) (the portable character set), are automatically included in this class.

**lower** Define characters to be classified as lowercase letters.

In the POSIX locale, the 26 lowercase letters shall be included:

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

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the portable character set are automatically included in this class.

**alpha** Define characters to be classified as letters.

In the POSIX locale, all characters in the classes **upper** and **lower** shall be included.

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. Characters classified as either **upper** or **lower** are automatically included in this class.

**digit** Define the characters to be classified as numeric digits.

In the POSIX locale, only:

0 1 2 3 4 5 6 7 8 9

shall be included.

In a locale definition file, only the digits <zero>, <one>, <two>, <three>, <four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in contiguous ascending sequence by numerical value. The digits <zero> to <nine> of the portable character set are automatically included in this class.

4128	<b>alnum</b>	Define characters to be classified as letters and numeric digits. Only the
4129		characters specified for the <b>alpha</b> and <b>digit</b> keywords shall be specified.
4130		Characters specified for the keywords <b>alpha</b> and <b>digit</b> are automatically
4131		included in this class.
4132	<b>space</b>	Define characters to be classified as white-space characters.
4133		In the POSIX locale, at a minimum, the <space>, <form-feed>, <newline>,
4134		<carriage-return>, <tab>, and <vertical-tab> shall be included.
4135		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4136		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>graph</b> , or <b>xdigit</b> shall be specified. The <space>, <form-
4137		feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
4138		character set, and any characters included in the class <b>blank</b> are automatically
4139		included in this class.
4140	<b>cntrl</b>	Define characters to be classified as control characters.
4141		In the POSIX locale, no characters in classes <b>alpha</b> or <b>print</b> shall be included.
4142		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4143		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>punct</b> , <b>graph</b> , <b>print</b> , or <b>xdigit</b> shall be specified.
4144	<b>punct</b>	Define characters to be classified as punctuation characters.
4145		In the POSIX locale, neither the <space> nor any characters in classes <b>alpha</b> ,
4146		<b>digit</b> , or <b>cntrl</b> shall be included.
4147		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4148		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>cntrl</b> , <b>xdigit</b> , or as the <space> shall be specified.
4149	<b>graph</b>	Define characters to be classified as printable characters, not including the
4150		<space>.
4151		In the POSIX locale, all characters in classes <b>alpha</b> , <b>digit</b> , and <b>punct</b> shall be
4152		included; no characters in class <b>cntrl</b> shall be included.
4153		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> ,
4154		<b>alpha</b> , <b>digit</b> , <b>xdigit</b> , and <b>punct</b> are automatically included in this class. No
4155		character specified for the keyword <b>cntrl</b> shall be specified.
4156	<b>print</b>	Define characters to be classified as printable characters, including the
4157		<space>.
4158		In the POSIX locale, all characters in class <b>graph</b> shall be included; no
4159		characters in class <b>cntrl</b> shall be included.
4160		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> ,
4161		<b>alpha</b> , <b>digit</b> , <b>xdigit</b> , <b>punct</b> , <b>graph</b> , and the <space> are automatically included
4162		in this class. No character specified for the keyword <b>cntrl</b> shall be specified.
4163	<b>xdigit</b>	Define the characters to be classified as hexadecimal digits.
4164		In the POSIX locale, only:
4165		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4166		shall be included.
4167		In a locale definition file, only the characters defined for the class <b>digit</b> shall be
4168		specified, in contiguous ascending sequence by numerical value, followed by
4169		one or more sets of six characters representing the hexadecimal digits 10 to 15

4170		inclusive, with each set in ascending order (for example, <A>, <B>, <C>, <D>, <E>, <F>, <a>, <b>, <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the portable character set are automatically included in this class.
4171		
4172		
4173		
4174	<b>blank</b>	Define characters to be classified as <blank>s.
4175		In the POSIX locale, only the <space> and <tab> shall be included.
4176		In a locale definition file, the <space> and <tab> are automatically included in this class.
4177		
4178	<b>charclass</b>	Define one or more locale-specific character class names as strings separated by semicolons. Each named character class can then be defined subsequently in the <i>LC_CTYPE</i> definition. A character class name shall consist of at least one and at most {CHARCLASS_NAME_MAX} bytes of alphanumeric characters from the portable filename character set. The first character of a character class name shall not be a digit. The name shall not match any of the <i>LC_CTYPE</i> keywords defined in this volume of IEEE Std 1003.1-2001. Future revisions of IEEE Std 1003.1-2001 will not specify any <i>LC_CTYPE</i> keywords containing uppercase letters.
4179		
4180		
4181		
4182		
4183		
4184		
4185		
4186		
4187	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific character class. In the POSIX locale, locale-specific named character classes need not exist.
4188		
4189		
4190		If a class name is defined by a <b>charclass</b> keyword, but no characters are subsequently assigned to it, this is not an error; it represents a class without any characters belonging to it.
4191		
4192		
4193		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i> function, in regular expression and shell pattern-matching bracket expressions, and by the <i>tr</i> command.
4194		
4195		
4196	<b>toupper</b>	Define the mapping of lowercase letters to uppercase letters.
4197		In the POSIX locale, at a minimum, the 26 lowercase characters:
4198		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
4199		shall be mapped to the corresponding 26 uppercase characters:
4200		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
4201		In a locale definition file, the operand shall consist of character pairs, separated by semicolons. The characters in each character pair shall be separated by a comma and the pair enclosed by parentheses. The first character in each pair is the lowercase letter, the second the corresponding uppercase letter. Only characters specified for the keywords <b>lower</b> and <b>upper</b> shall be specified. The lowercase letters <a> to <z>, and their corresponding uppercase letters <A> to <Z>, of the portable character set are automatically included in this mapping, but only when the <b>toupper</b> keyword is omitted from the locale definition.
4202		
4203		
4204		
4205		
4206		
4207		
4208		
4209		
4210	<b>tolower</b>	Define the mapping of uppercase letters to lowercase letters.
4211		In the POSIX locale, at a minimum, the 26 uppercase characters:
4212		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

shall be mapped to the corresponding 26 lowercase characters:

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

In a locale definition file, the operand shall consist of character pairs, separated by semicolons. The characters in each character pair shall be separated by a comma and the pair enclosed by parentheses. The first character in each pair is the uppercase letter, the second the corresponding lowercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from the locale definition, the mapping is the reverse mapping of the one specified for **toupper**.

The following table shows the character class combinations allowed:

**Table 7-1** Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
<b>upper</b>	—	—	A	x	x	x	x	A	A	—	x
<b>lower</b>	—	—	A	x	x	x	x	A	A	—	x
<b>alpha</b>	—	—	—	x	x	x	x	A	A	—	x
<b>digit</b>	x	x	x	—	x	x	x	A	A	A	x
<b>space</b>	x	x	x	x	—	—	*	*	*	x	—
<b>cntrl</b>	x	x	x	x	—	—	x	x	x	x	—
<b>punct</b>	x	x	x	x	—	x	—	A	A	x	—
<b>graph</b>	—	—	—	—	—	x	—	—	A	—	—
<b>print</b>	—	—	—	—	—	x	—	—	—	—	—
<b>xdigit</b>	—	—	—	—	x	x	x	A	A	—	x
<b>blank</b>	x	x	x	x	A	—	*	*	*	x	—

**Notes:**

1. Explanation of codes:

A Automatically included; see text.

— Permitted.

x Mutually-exclusive.

\* See note 2.

2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

### 7.3.1.1 LC\_CTYPE Category in the POSIX Locale

The character classifications for the POSIX locale follow; the code listing depicts the *localedef* input, and the table represents the same information, sorted by character.

```
LC_CTYPE
# The following is the POSIX locale LC_CTYPE.
# "alpha" is by default "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by default "alnum", "punct", and the <space>
# "graph" is by default "alnum" and "punct"
#
upper      <A>; <B>; <C>; <D>; <E>; <F>; <G>; <H>; <I>; <J>; <K>; <L>; <M>; \
```



```

4257          <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4258      #
4259      lower  <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4260            <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4261      #
4262      digit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4263            <seven>;<eight>;<nine>
4264      #
4265      space  <tab>;<newline>;<vertical-tab>;<form-feed>;\
4266            <carriage-return>;<space>
4267      #
4268      cntrl  <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4269            <form-feed>;<carriage-return>;\
4270            <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4271            <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4272            <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4273            <IS1>;<DEL>
4274      #
4275      punct  <exclamation-mark>;<quotation-mark>;<number-sign>;\
4276            <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4277            <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4278            <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4279            <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4280            <greater-than-sign>;<question-mark>;<commercial-at>;\
4281            <left-square-bracket>;<backslash>;<right-square-bracket>;\
4282            <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4283            <vertical-line>;<right-curly-bracket>;<tilde>
4284      #
4285      xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4286            <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4287      #
4288      blank  <space>;<tab>
4289      #
4290      toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4291            (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4292            (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4293            (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4294            (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4295      #
4296      tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4297            (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
4298            (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
4299            (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
4300            (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
4301      END LC_CTYPE

```

4302  
4303  
4304  
4305  
4306  
4307  
4308  
4309  
4310  
4311  
4312  
4313  
4314  
4315  
4316  
4317  
4318  
4319  
4320  
4321  
4322  
4323  
4324  
4325  
4326  
4327  
4328  
4329  
4330  
4331  
4332  
4333  
4334  
4335  
4336  
4337  
4338  
4339  
4340  
4341  
4342  
4343  
4344  
4345  
4346  
4347  
4348  
4349  
4350

Symbolic Name	Other Case	Character Classes
<NUL>		cntrl
<SOH>		cntrl
<STX>		cntrl
<ETX>		cntrl
<EOT>		cntrl
<ENQ>		cntrl
<ACK>		cntrl
<alert>		cntrl
<backspace>		cntrl
<tab>		cntrl, space, blank
<newline>		cntrl, space
<vertical-tab>		cntrl, space
<form-feed>		cntrl, space
<carriage-return>		cntrl, space
<SO>		cntrl
<SI>		cntrl
<DLE>		cntrl
<DC1>		cntrl
<DC2>		cntrl
<DC3>		cntrl
<DC4>		cntrl
<NAK>		cntrl
<SYN>		cntrl
<ETB>		cntrl
<CAN>		cntrl
<EM>		cntrl
<SUB>		cntrl
<ESC>		cntrl
<IS4>		cntrl
<IS3>		cntrl
<IS2>		cntrl
<IS1>		cntrl
<space>		space, print, blank
<exclamation-mark>		punct, print, graph
<quotation-mark>		punct, print, graph
<number-sign>		punct, print, graph
<dollar-sign>		punct, print, graph
<percent-sign>		punct, print, graph
<ampersand>		punct, print, graph
<apostrophe>		punct, print, graph
<left-parenthesis>		punct, print, graph
<right-parenthesis>		punct, print, graph
<asterisk>		punct, print, graph
<plus-sign>		punct, print, graph
<comma>		punct, print, graph
<hyphen>		punct, print, graph
<period>		punct, print, graph

4351			
4352	<b>Symbolic Name</b>	<b>Other Case</b>	<b>Character Classes</b>
4353	<slash>		punct, print, graph
4354	<zero>		digit, xdigit, print, graph
4355	<one>		digit, xdigit, print, graph
4356	<two>		digit, xdigit, print, graph
4357	<three>		digit, xdigit, print, graph
4358	<four>		digit, xdigit, print, graph
4359	<five>		digit, xdigit, print, graph
4360	<six>		digit, xdigit, print, graph
4361	<seven>		digit, xdigit, print, graph
4362	<eight>		digit, xdigit, print, graph
4363	<nine>		digit, xdigit, print, graph
4364	<colon>		punct, print, graph
4365	<semicolon>		punct, print, graph
4366	<less-than-sign>		punct, print, graph
4367	<equals-sign>		punct, print, graph
4368	<greater-than-sign>		punct, print, graph
4369	<question-mark>		punct, print, graph
4370	<commercial-at>		punct, print, graph
4371	<A>	<a>	upper, xdigit, alpha, print, graph
4372	<B>	<b>	upper, xdigit, alpha, print, graph
4373	<C>	<c>	upper, xdigit, alpha, print, graph
4374	<D>	<d>	upper, xdigit, alpha, print, graph
4375	<E>	<e>	upper, xdigit, alpha, print, graph
4376	<F>	<f>	upper, xdigit, alpha, print, graph
4377	<G>	<g>	upper, alpha, print, graph
4378	<H>	<h>	upper, alpha, print, graph
4379	<I>	<i>	upper, alpha, print, graph
4380	<J>	<j>	upper, alpha, print, graph
4381	<K>	<k>	upper, alpha, print, graph
4382	<L>	<l>	upper, alpha, print, graph
4383	<M>	<m>	upper, alpha, print, graph
4384	<N>	<n>	upper, alpha, print, graph
4385	<O>	<o>	upper, alpha, print, graph
4386	<P>	<p>	upper, alpha, print, graph
4387	<Q>	<q>	upper, alpha, print, graph
4388	<R>	<r>	upper, alpha, print, graph
4389	<S>	<s>	upper, alpha, print, graph
4390	<T>	<t>	upper, alpha, print, graph
4391	<U>	<u>	upper, alpha, print, graph
4392	<V>	<v>	upper, alpha, print, graph
4393	<W>	<w>	upper, alpha, print, graph
4394	<X>	<x>	upper, alpha, print, graph
4395	<Y>	<y>	upper, alpha, print, graph
4396	<Z>	<z>	upper, alpha, print, graph
4397	<left-square-bracket>		punct, print, graph
4398	<backslash>		punct, print, graph
4399	<right-square-bracket>		punct, print, graph

4400  
4401  
4402  
4403  
4404  
4405  
4406  
4407  
4408  
4409  
4410  
4411  
4412  
4413  
4414  
4415  
4416  
4417  
4418  
4419  
4420  
4421  
4422  
4423  
4424  
4425  
4426  
4427  
4428  
4429  
4430  
4431  
4432  
4433  
4434  
4435

Symbolic Name	Other Case	Character Classes
<circumflex>		punct, print, graph
<underscore>		punct, print, graph
<grave-accent>		punct, print, graph
<a>	<A>	lower, xdigit, alpha, print, graph
<b>	<B>	lower, xdigit, alpha, print, graph
<c>	<C>	lower, xdigit, alpha, print, graph
<d>	<D>	lower, xdigit, alpha, print, graph
<e>	<E>	lower, xdigit, alpha, print, graph
<f>	<F>	lower, xdigit, alpha, print, graph
<g>	<G>	lower, alpha, print, graph
<h>	<H>	lower, alpha, print, graph
<i>	<I>	lower, alpha, print, graph
<j>	<J>	lower, alpha, print, graph
<k>	<K>	lower, alpha, print, graph
<l>	<L>	lower, alpha, print, graph
<m>	<M>	lower, alpha, print, graph
<n>	<N>	lower, alpha, print, graph
<o>	<O>	lower, alpha, print, graph
<p>	<P>	lower, alpha, print, graph
<q>	<Q>	lower, alpha, print, graph
<r>	<R>	lower, alpha, print, graph
<s>	<S>	lower, alpha, print, graph
<t>	<T>	lower, alpha, print, graph
<u>	<U>	lower, alpha, print, graph
<v>	<V>	lower, alpha, print, graph
<w>	<W>	lower, alpha, print, graph
<x>	<X>	lower, alpha, print, graph
<y>	<Y>	lower, alpha, print, graph
<z>	<Z>	lower, alpha, print, graph
<left-curly-bracket>		punct, print, graph
<vertical-line>		punct, print, graph
<right-curly-bracket>		punct, print, graph
<tilde>		punct, print, graph
<DEL>		cntrl

### 4436 7.3.2 LC\_COLLATE

4437 The *LC\_COLLATE* category provides a collation sequence definition for numerous utilities in the  
4438 Shell and Utilities volume of IEEE Std 1003.1-2001 (*sort*, *uniq*, and so on), regular expression  
4439 matching (see Chapter 9 (on page 169)), and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()*  
4440 functions in the System Interfaces volume of IEEE Std 1003.1-2001.

4441 A collation sequence definition shall define the relative order between collating elements  
4442 (characters and multi-character collating elements) in the locale. This order is expressed in terms  
4443 of collation values; that is, by assigning each element one or more collation values (also known  
4444 as collation weights). This does not imply that implementations shall assign such values, but  
4445 that ordering of strings using the resultant collation definition in the locale behaves as if such  
4446 assignment is done and used in the collation process. At least the following capabilities are  
4447 provided:

- 4448 1. **Multi-character collating elements.** Specification of multi-character collating elements  
4449 (that is, sequences of two or more characters to be collated as an entity).

2. **User-defined ordering of collating elements.** Each collating element shall be assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicitly specified, also as the collation weight to be used in sorting.
3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or more (up to the limit {COLL\_WEIGHTS\_MAX}, as defined in <limits.h>) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.
4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
5. **Equivalence class definition.** Two or more collating elements have the same collation value (primary weight).
6. **Ordering by weights.** When two strings are compared to determine their relative order, the two strings are first broken up into a series of collating elements; the elements in each successive pair of elements are then compared according to the relative primary weights for the elements. If equal, and more than one weight has been assigned, then the pairs of collating elements are re-compared according to the relative subsequent weights, until either a pair of collating elements compare unequal or the weights are exhausted.

The following keywords shall be recognized in a collation sequence definition. They are described in detail in the following sections.

<b>copy</b>	Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.
<b>collating-element</b>	Define a collating-element symbol representing a multi-character collating element. This keyword is optional.
<b>collating-symbol</b>	Define a collating symbol for use in collation order statements. This keyword is optional.
<b>order_start</b>	Define collation rules. This statement shall be followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
<b>order_end</b>	Specify the end of the collation-order statements.

### 7.3.2.1 The collating-element Keyword

In addition to the collating elements in the character set, the **collating-element** keyword can be used to define multi-character collating elements. The syntax is as follows:

```
"collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

The <collating-symbol> operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <collating-element> defined via this keyword is only recognized with the LC\_COLLATE category.

For example:

```
collating-element <ch> from "<c><h>"
collating-element <e-acute> from "<acute><e>"
collating-element <ll> from "ll"
```

4492 7.3.2.2 *The collating-symbol Keyword*

4493 This keyword shall be used to define symbols for use in collation sequence statements; that is,  
 4494 between the **order\_start** and the **order\_end** keywords. The syntax is as follows:

4495 "collating-symbol %s\n", <collating-symbol>

4496 The <collating-symbol> shall be a symbolic name, enclosed between angle brackets ('<' and  
 4497 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any  
 4498 other symbolic name defined in this collation definition. A <collating-symbol> defined via this  
 4499 keyword is only recognized within the *LC\_COLLATE* category.

4500 For example:

4501 collating-symbol <UPPER\_CASE>

4502 collating-symbol <HIGH>

4503 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative  
 4504 position in the character order sequence. While such a symbolic name does not represent any  
 4505 collating element, it can be used as a weight.

4506 7.3.2.3 *The order\_start Keyword*

4507 The **order\_start** keyword shall precede collation order entries and also define the number of  
 4508 weights for this collation sequence definition and other collation rules. The syntax is as follows:

4509 "order\_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...

4510 The operands to the **order\_start** keyword are optional. If present, the operands define rules to be  
 4511 applied when strings are compared. The number of operands define how many weights each  
 4512 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the  
 4513 first operand defines rules to be applied when comparing strings using the first (primary)  
 4514 weight; the second when comparing strings using the second weight, and so on. Operands shall  
 4515 be separated by semicolons (';'). Each operand shall consist of one or more collation  
 4516 directives, separated by commas (','),. If the number of operands exceeds the  
 4517 {COLL\_WEIGHTS\_MAX} limit, the utility shall issue a warning message. The following  
 4518 directives shall be supported:

4519 **forward** Specifies that comparison operations for the weight level shall proceed from start  
 4520 of string towards the end of string.

4521 **backward** Specifies that comparison operations for the weight level shall proceed from end of  
 4522 string towards the beginning of string.

4523 **position** Specifies that comparison operations for the weight level shall consider the relative  
 4524 position of elements in the strings not subject to **IGNORE**. The string containing  
 4525 an element not subject to **IGNORE** after the fewest collating elements subject to  
 4526 **IGNORE** from the start of the compare shall collate first. If both strings contain a  
 4527 character not subject to **IGNORE** in the same relative position, the collating values  
 4528 assigned to the elements shall determine the ordering. In case of equality,  
 4529 subsequent characters not subject to **IGNORE** shall be considered in the same  
 4530 manner.

4531 The directives **forward** and **backward** are mutually-exclusive.

4532 If no operands are specified, a single **forward** operand shall be assumed.

4533 For example:

4534       order\_start       forward;backward

#### 4535 7.3.2.4 Collation Order

4536 The **order\_start** keyword shall be followed by collating identifier entries. The syntax for the  
4537 collating element entries is as follows:

4538       "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...

4539 Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section  
4540 7.3 (on page 124)), a <collating-element>, a <collating-symbol>, an ellipsis, or the special symbol  
4541 **UNDEFINED**. The order in which collating elements are specified determines the character  
4542 order sequence, such that each collating element shall compare less than the elements following  
4543 it.

4544 A <collating-element> shall be used to specify multi-character collating elements, and indicates  
4545 that the character sequence specified via the <collating-element> is to be collated as a unit and in  
4546 the relative order specified by its place.

4547 A <collating-symbol> can be used to define a position in the relative order for use in weights. No  
4548 weights shall be specified with a <collating-symbol>.

4549 The ellipsis symbol specifies that a sequence of characters shall collate according to their  
4550 encoded character values. It shall be interpreted as indicating that all characters with a coded  
4551 character set value higher than the value of the character in the preceding line, and lower than  
4552 the coded character set value for the character in the following line, in the current coded  
4553 character set, shall be placed in the character collation order between the previous and the  
4554 following character in ascending order according to their coded character set values. An initial  
4555 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing  
4556 ellipsis as if the following line specified the highest coded character set value in the current  
4557 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do  
4558 not specify characters in the current coded character set. The use of the ellipsis symbol ties the  
4559 definition to a specific coded character set and may preclude the definition from being portable  
4560 between implementations.

4561 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not  
4562 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character  
4563 collation order at the point indicated by the symbol, and in ascending order according to their  
4564 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded  
4565 character set contains characters not specified in this section, the utility shall issue a warning  
4566 message and place such characters at the end of the character collation order.

4567 The optional operands for each collation-element shall be used to define the primary, secondary,  
4568 or subsequent weights for the collating element. The first operand specifies the relative primary  
4569 weight, the second the relative secondary weight, and so on. Two or more collation-elements can  
4570 be assigned the same weight; they belong to the same "equivalence class" if they have the same  
4571 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**  
4572 are removed, unless the **position** collation directive is specified for the corresponding level with  
4573 the **order\_start** keyword. Then each successive pair of elements shall be compared according to  
4574 the relative weights for the elements. If the two strings compare equal, the process shall be  
4575 repeated for the next weight level, up to the limit {COLL\_WEIGHTS\_MAX}.

4576 Weights shall be expressed as characters (in any of the forms specified in Section 7.3 (on page  
4577 124)), <collating-symbol>s, <collating-element>s, an ellipsis, or the special symbol **IGNORE**. A  
4578 single character, a <collating-symbol>, or a <collating-element> shall represent the relative position

in the character collating sequence of the character or symbol, rather than the character or characters themselves. Thus, rather than assigning absolute values to weights, a particular weight is expressed using the relative order value assigned to a collating element based on its order in the character collation sequence.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. For example, if the `<eszet>` is given the string "`<s><s>`" as a weight, comparisons are performed as if all occurrences of the `<eszet>` are replaced by "`<s><s>`" (assuming that "`<s>`" has the collating weight "`<s>`"). If it is necessary to define `<eszet>` and "`<s><s>`" as an equivalence class, then a collating element must be defined for the string "`ss`".

All characters specified via an ellipsis shall by default be assigned unique weights, equal to the relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special symbol shall by default be assigned the same primary weight (that is, they belong to the same equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence shall have unique weights, equal to the relative order of their character in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

The special keyword **IGNORE** as a weight shall indicate that when strings are compared using the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that is, as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are subject to **IGNORE** in their primary weight form an equivalence class.

An empty operand shall be interpreted as the collating element itself.

For example, the order statement:

```
<a>      <a> i <a>
```

is equal to:

```
<a>
```

An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be interpreted as the value of each character defined by the ellipsis.

The collation order as defined in this section affects the interpretation of bracket expressions in regular expressions (see Section 9.3.5 (on page 172)).

For example:



```

4610      order_start    forward;backward
4611      UNDEFINED      IGNORE;IGNORE
4612      <LOW>
4613      <space>         <LOW>;<space>
4614      ...             <LOW>;...
4615      <a>              <a>;<a>
4616      <a-acute>        <a>;<a-acute>
4617      <a-grave>        <a>;<a-grave>
4618      <A>              <a>;<A>
4619      <A-acute>        <a>;<A-acute>
4620      <A-grave>        <a>;<A-grave>
4621      <ch>             <ch>;<ch>
4622      <Ch>             <ch>;<Ch>
4623      <s>              <s>;<s>
4624      <eszet>         "<s><s>";"<eszet><eszet>"
4625      order_end

```

4626 This example is interpreted as follows:

- 4627 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or  
4628 via the ellipsis) shall be ignored for collation purposes.
- 4629 2. All characters between <space> and 'a' shall have the same primary equivalence class  
4630 and individual secondary weights based on their ordinal encoded values.
- 4631 3. All characters based on the uppercase or lowercase character 'a' belong to the same  
4632 primary equivalence class.
- 4633 4. The multi-character collating element <ch> is represented by the collating symbol <ch>  
4634 and belongs to the same primary equivalence class as the multi-character collating element  
4635 <Ch>.

#### 4636 7.3.2.5 The order\_end Keyword

4637 The collating order entries shall be terminated with an **order\_end** keyword.

#### 4638 7.3.2.6 LC\_COLLATE Category in the POSIX Locale

4639 The collation sequence definition of the POSIX locale follows; the code listing depicts the  
4640 *localedef* input.

```

4641 LC_COLLATE
4642 # This is the POSIX locale definition for the LC_COLLATE category.
4643 # The order is the same as in the ASCII codeset.
4644 order_start forward
4645 <NUL>
4646 <SOH>
4647 <STX>
4648 <ETX>
4649 <EOT>
4650 <ENQ>
4651 <ACK>
4652 <alert>
4653 <backspace>
4654 <tab>
4655 <newline>

```

4656 <vertical-tab>  
 4657 <form-feed>  
 4658 <carriage-return>  
 4659 <SO>  
 4660 <SI>  
 4661 <DLE>  
 4662 <DC1>  
 4663 <DC2>  
 4664 <DC3>  
 4665 <DC4>  
 4666 <NAK>  
 4667 <SYN>  
 4668 <ETB>  
 4669 <CAN>  
 4670 <EM>  
 4671 <SUB>  
 4672 <ESC>  
 4673 <IS4>  
 4674 <IS3>  
 4675 <IS2>  
 4676 <IS1>  
 4677 <space>  
 4678 <exclamation-mark>  
 4679 <quotation-mark>  
 4680 <number-sign>  
 4681 <dollar-sign>  
 4682 <percent-sign>  
 4683 <ampersand>  
 4684 <apostrophe>  
 4685 <left-parenthesis>  
 4686 <right-parenthesis>  
 4687 <asterisk>  
 4688 <plus-sign>  
 4689 <comma>  
 4690 <hyphen>  
 4691 <period>  
 4692 <slash>  
 4693 <zero>  
 4694 <one>  
 4695 <two>  
 4696 <three>  
 4697 <four>  
 4698 <five>  
 4699 <six>  
 4700 <seven>  
 4701 <eight>  
 4702 <nine>  
 4703 <colon>  
 4704 <semicolon>  
 4705 <less-than-sign>  
 4706 <equals-sign>  
 4707 <greater-than-sign>

4708	<question-mark>
4709	<commercial-at>
4710	<A>
4711	<B>
4712	<C>
4713	<D>
4714	<E>
4715	<F>
4716	<G>
4717	<H>
4718	<I>
4719	<J>
4720	<K>
4721	<L>
4722	<M>
4723	<N>
4724	<O>
4725	<P>
4726	<Q>
4727	<R>
4728	<S>
4729	<T>
4730	<U>
4731	<V>
4732	<W>
4733	<X>
4734	<Y>
4735	<Z>
4736	<left-square-bracket>
4737	<backslash>
4738	<right-square-bracket>
4739	<circumflex>
4740	<underscore>
4741	<grave-accent>
4742	<a>
4743	<b>
4744	<c>
4745	<d>
4746	<e>
4747	<f>
4748	<g>
4749	<h>
4750	<i>
4751	<j>
4752	<k>
4753	<l>
4754	<m>
4755	<n>
4756	<o>
4757	<p>
4758	<q>
4759	<r>

```

4760      <s>
4761      <t>
4762      <u>
4763      <v>
4764      <w>
4765      <x>
4766      <y>
4767      <z>
4768      <left-curly-bracket>
4769      <vertical-line>
4770      <right-curly-bracket>
4771      <tilde>
4772      <DEL>
4773      order_end
4774      #
4775      END LC_COLLATE

```

### 4776 7.3.3 LC\_MONETARY

4777 The *LC\_MONETARY* category shall define the rules and symbols that are used to format  
 4778 monetary numeric information.

4779 XSI This information is available through the *localeconv()* function and is used by the *strfmon()*  
 4780 function.

4781 XSI Some of the information is also available in an alternative form via the *nl\_langinfo()* function  
 4782 (see CRNCYSTR in *<langinfo.h>*).

4783 The following items are defined in this category of the locale. The item names are the keywords  
 4784 recognized by the *localedef* utility when defining a locale. They are also similar to the member  
 4785 names of the *lconv* structure defined in *<locale.h>*; see *<locale.h>* for the exact symbols in the  
 4786 header. The *localeconv()* function returns {CHAR\_MAX} for unspecified integer items and the  
 4787 empty string ( " ") for unspecified or size zero string items.

4788 In a locale definition file, the operands are strings, formatted as indicated by the grammar in  
 4789 Section 7.4 (on page 153). For some keywords, the strings can contain only integers. Keywords  
 4790 that are not provided, string values set to the empty string ( " "), or integer keywords set to -1,  
 4791 are used to indicate that the value is not available in the locale. The following keywords shall be  
 4792 recognized:

4793 **copy** Specify the name of an existing locale which shall be used as the  
 4794 definition of this category. If this keyword is specified, no other keyword  
 4795 shall be specified.

4796 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4797 **int\_curr\_symbol** The international currency symbol. The operand shall be a four-character  
 4798 string, with the first three characters containing the alphabetic  
 4799 international currency symbol. The international currency symbol should  
 4800 be chosen in accordance with those specified in the ISO 4217 standard.  
 4801 The fourth character shall be the character used to separate the  
 4802 international currency symbol from the monetary quantity.

4803 **currency\_symbol** The string that shall be used as the local currency symbol.

4804 **mon\_decimal\_point** The operand is a string containing the symbol that shall be used as the  
 4805 decimal delimiter (radix character) in monetary formatted quantities.

4806	<b>mon_thousands_sep</b>	The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.	
4807			
4808			
4809	<b>mon_grouping</b>	Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not -1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is -1, then no further grouping shall be performed.	
4810			
4811			
4812			
4813			
4814			
4815			
4816			
4817	<b>positive_sign</b>	A string that shall be used to indicate a non-negative-valued formatted monetary quantity.	
4818			
4819	<b>negative_sign</b>	A string that shall be used to indicate a negative-valued formatted monetary quantity.	
4820			
4821	<b>int_frac_digits</b>	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using <b>int_curr_symbol</b> .	
4822			
4823			
4824	<b>frac_digits</b>	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using <b>currency_symbol</b> .	
4825			
4826			
4827	<b>p_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.	
4828			
4829			
4830	<b>p_sep_by_space</b>	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the value for a non-negative formatted monetary quantity.	2
4831			2
4832		The values of <b>p_sep_by_space</b> , <b>n_sep_by_space</b> , <b>int_p_sep_by_space</b> , and <b>int_n_sep_by_space</b> are interpreted according to the following:	2
4833			2
4834		0 No space separates the currency symbol and value.	2
4835		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
4836			2
4837			2
4838		2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a space separates the sign string from the value.	2
4839			2
4840	<b>n_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.	
4841			
4842			
4843	<b>n_sep_by_space</b>	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the value for a negative formatted monetary quantity.	2
4844			2
4845	<b>p_sign_posn</b>	An integer set to a value indicating the positioning of the <b>positive_sign</b> for a monetary quantity with a non-negative value. The following integer values shall be recognized for <b>int_n_sign_posn</b> , <b>int_p_sign_posn</b> , <b>n_sign_posn</b> , and <b>p_sign_posn</b> :	
4846			
4847			
4848			
4849		0 Parentheses enclose the quantity and the <b>currency_symbol</b> .	

4850		1	The sign string precedes the quantity and the <b>currency_symbol</b> .	
4851		2	The sign string succeeds the quantity and the <b>currency_symbol</b> .	
4852		3	The sign string precedes the <b>currency_symbol</b> .	
4853		4	The sign string succeeds the <b>currency_symbol</b> .	
4854	<b>n_sign_posn</b>		An integer set to a value indicating the positioning of the <b>negative_sign</b>	
4855			for a negative formatted monetary quantity.	
4856	<b>int_p_cs_precedes</b>		An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a	
4857			monetary quantity with a non-negative value, and set to 0 if the symbol	
4858			succeeds the value.	
4859	<b>int_n_cs_precedes</b>		An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a	
4860			monetary quantity with a negative value, and set to 0 if the symbol	
4861			succeeds the value.	
4862	<b>int_p_sep_by_space</b>		Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign	2
4863			string, and the value for a non-negative internationally formatted	2
4864			monetary quantity.	2
4865	<b>int_n_sep_by_space</b>		Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign	2
4866			string, and the value for a negative internationally formatted monetary	2
4867			quantity.	2
4868	<b>int_p_sign_posn</b>		An integer set to a value indicating the positioning of the <b>positive_sign</b>	
4869			for a positive monetary quantity formatted with the international format.	
4870	<b>int_n_sign_posn</b>		An integer set to a value indicating the positioning of the <b>negative_sign</b>	
4871			for a negative monetary quantity formatted with the international format.	
4872	<b>7.3.3.1 LC_MONETARY Category in the POSIX Locale</b>			
4873			The monetary formatting definitions for the POSIX locale follow; the code listing depicting the	
4874	XSI		<i>localedef</i> input, the table representing the same information with the addition of <i>localeconv()</i> and	
4875			<i>nl_langinfo()</i> formats. All values are unspecified in the POSIX locale.	
4876	LC_MONETARY			
4877	# This is the POSIX locale definition for			
4878	# the LC_MONETARY category.			
4879	#			
4880	int_curr_symbol	" "		
4881	currency_symbol	" "		
4882	mon_decimal_point	" "		
4883	mon_thousands_sep	" "		
4884	mon_grouping	-1		
4885	positive_sign	" "		
4886	negative_sign	" "		
4887	int_frac_digits	-1		
4888	frac_digits	-1		
4889	p_cs_precedes	-1		
4890	p_sep_by_space	-1		
4891	n_cs_precedes	-1		
4892	n_sep_by_space	-1		
4893	p_sign_posn	-1		
4894	n_sign_posn	-1		

```

4895      int_p_cs_precedes      -1      1
4896      int_p_sep_by_space     -1      1
4897      int_n_cs_precedes      -1      1
4898      int_n_sep_by_space     -1      1
4899      int_p_sign_posn        -1      1
4900      int_n_sign_posn        -1      1
4901      #
4902      END LC_MONETARY

```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value	
int_curr_symbol	—	N/A	" "	" "	
currency_symbol	CRNCYSTR	N/A	" "	" "	
mon_decimal_point	—	N/A	" "	" "	
mon_thousands_sep	—	N/A	" "	" "	
mon_grouping	—	N/A	" "	-1	
positive_sign	—	N/A	" "	" "	
negative_sign	—	N/A	" "	" "	
int_frac_digits	—	N/A	{CHAR_MAX}	-1	
frac_digits	—	N/A	{CHAR_MAX}	-1	
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1	
p_sep_by_space	—	N/A	{CHAR_MAX}	-1	
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1	
n_sep_by_space	—	N/A	{CHAR_MAX}	-1	
p_sign_posn	—	N/A	{CHAR_MAX}	-1	
n_sign_posn	—	N/A	{CHAR_MAX}	-1	
int_p_cs_precedes	—	N/A	{CHAR_MAX}	-1	1
int_p_sep_by_space	—	N/A	{CHAR_MAX}	-1	1
int_n_cs_precedes	—	N/A	{CHAR_MAX}	-1	1
int_n_sep_by_space	—	N/A	{CHAR_MAX}	-1	1
int_p_sign_posn	—	N/A	{CHAR_MAX}	-1	1
int_n_sign_posn	—	N/A	{CHAR_MAX}	-1	1

4926 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.  
4927 The entry N/A indicates that the value is not available in the POSIX locale.

#### 4928 7.3.4 LC\_NUMERIC

4929 The *LC\_NUMERIC* category shall define the rules and symbols that are used to format non-  
4930 monetary numeric information. This information is available through the *localeconv()* function.

4931 XSI Some of the information is also available in an alternative form via the *nl\_langinfo()* function.

4932 The following items are defined in this category of the locale. The item names are the keywords  
4933 recognized by the *localedef* utility when defining a locale. They are also similar to the member  
4934 names of the **lconv** structure defined in **<locale.h>**; see **<locale.h>** for the exact symbols in the  
4935 header. The *localeconv()* function returns {CHAR\_MAX} for unspecified integer items and the  
4936 empty string ( " ") for unspecified or size zero string items.

4937 In a locale definition file, the operands are strings, formatted as indicated by the grammar in  
4938 Section 7.4 (on page 153). For some keywords, the strings can only contain integers. Keywords  
4939 that are not provided, string values set to the empty string ( " "), or integer keywords set to -1,  
4940 shall be used to indicate that the value is not available in the locale. The following keywords  
4941 shall be recognized:

4942       **copy**           Specify the name of an existing locale which shall be used as the definition of  
 4943                       this category. If this keyword is specified, no other keyword shall be specified.

4944               **Note:**       This is a *localedef* utility keyword, unavailable through *localeconv()*.

4945       **decimal\_point**   The operand is a string containing the symbol that shall be used as the  
 4946                       decimal delimiter (radix character) in numeric, non-monetary formatted  
 4947                       quantities. This keyword cannot be omitted and cannot be set to the empty  
 4948                       string. In contexts where standards limit the **decimal\_point** to a single byte,  
 4949                       the result of specifying a multi-byte operand shall be unspecified.

4950       **thousands\_sep**   The operand is a string containing the symbol that shall be used as a separator  
 4951                       for groups of digits to the left of the decimal delimiter in numeric, non-  
 4952                       monetary formatted monetary quantities. In contexts where standards limit  
 4953                       the **thousands\_sep** to a single byte, the result of specifying a multi-byte  
 4954                       operand shall be unspecified.

4955       **grouping**        Define the size of each group of digits in formatted non-monetary quantities.  
 4956                       The operand is a sequence of integers separated by semicolons. Each integer  
 4957                       specifies the number of digits in each group, with the initial integer defining  
 4958                       the size of the group immediately preceding the decimal delimiter, and the  
 4959                       following integers defining the preceding groups. If the last integer is not -1,  
 4960                       then the size of the previous group (if any) shall be repeatedly used for the  
 4961                       remainder of the digits. If the last integer is -1, then no further grouping shall  
 4962                       be performed.

#### 4963 7.3.4.1 LC\_NUMERIC Category in the POSIX Locale

4964       The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing  
 4965       depicting the *localedef* input, the table representing the same information with the addition of  
 4966       XSI *localeconv()* values, and *nl\_langinfo()* constants.

```

4967 LC_NUMERIC
4968 # This is the POSIX locale definition for
4969 # the LC_NUMERIC category.
4970 #
4971 decimal_point      "<period>"
4972 thousands_sep      " "
4973 grouping           -1
4974 #
4975 END LC_NUMERIC

```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
<b>decimal_point</b>	RADIXCHAR	"."	"."	"."
<b>thousands_sep</b>	THOUSEP	N/A	" "	" "
<b>grouping</b>	—	N/A	" "	-1

4981       XSI       In the preceding table, the **langinfo Constant** column represents an XSI-conforming extension.  
 4982                       The entry N/A indicates that the value is not available in the POSIX locale.



4983 **7.3.5 LC\_TIME**

4984 The *LC\_TIME* category shall define the interpretation of the conversion specifications supported  
 4985 XSI by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and  
 4986 *nl\_langinfo()* functions. Since the interfaces for C-language access and locale definition differ  
 4987 significantly, they are described separately.

4988 **7.3.5.1 LC\_TIME Locale Definition**

4989 In a locale definition, the following mandatory keywords shall be recognized:

4990	<b>copy</b>	Specify the name of an existing locale which shall be used as the definition of
4991		this category. If this keyword is specified, no other keyword shall be specified.
4992	<b>abday</b>	Define the abbreviated weekday names, corresponding to the %a conversion
4993		specification (conversion specification in the <i>strftime()</i> , <i>wcsftime()</i> , and
4994		<i>strptime()</i> functions). The operand shall consist of seven semicolon-separated
4995		strings, each surrounded by double-quotes. The first string shall be the
4996		abbreviated name of the day corresponding to Sunday, the second the
4997		abbreviated name of the day corresponding to Monday, and so on.
4998	<b>day</b>	Define the full weekday names, corresponding to the %A conversion
4999		specification. The operand shall consist of seven semicolon-separated strings,
5000		each surrounded by double-quotes. The first string is the full name of the day
5001		corresponding to Sunday, the second the full name of the day corresponding
5002		to Monday, and so on.
5003	<b>abmon</b>	Define the abbreviated month names, corresponding to the %b conversion
5004		specification. The operand shall consist of twelve semicolon-separated strings,
5005		each surrounded by double-quotes. The first string shall be the abbreviated
5006		name of the first month of the year (January), the second the abbreviated
5007		name of the second month, and so on.
5008	<b>mon</b>	Define the full month names, corresponding to the %B conversion
5009		specification. The operand shall consist of twelve semicolon-separated strings,
5010		each surrounded by double-quotes. The first string shall be the full name of
5011		the first month of the year (January), the second the full name of the second
5012		month, and so on.
5013	<b>d_t_fmt</b>	Define the appropriate date and time representation, corresponding to the %c
5014		conversion specification. The operand shall consist of a string containing any
5015		combination of characters and conversion specifications. In addition, the
5016		string can contain escape sequences defined in the table in Table 5-1 (on page
5017		110) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').
5018	<b>d_fmt</b>	Define the appropriate date representation, corresponding to the %x
5019		conversion specification. The operand shall consist of a string containing any
5020		combination of characters and conversion specifications. In addition, the
5021		string can contain escape sequences defined in Table 5-1 (on page 110).
5022	<b>t_fmt</b>	Define the appropriate time representation, corresponding to the %X
5023		conversion specification. The operand shall consist of a string containing any
5024		combination of characters and conversion specifications. In addition, the
5025		string can contain escape sequences defined in Table 5-1 (on page 110).
5026	<b>am_pm</b>	Define the appropriate representation of the <i>ante-meridiem</i> and <i>post-meridiem</i>
5027		strings, corresponding to the %p conversion specification. The operand shall
5028		consist of two strings, separated by a semicolon, each surrounded by double-

5029		quotes. The first string shall represent the <i>ante-meridiem</i> designation, the last
5030		string the <i>post-meridiem</i> designation.
5031	<b>t_fmt_ampm</b>	Define the appropriate time representation in the 12-hour clock format with
5032		<b>am_pm</b> , corresponding to the %r conversion specification. The operand shall
5033		consist of a string and can contain any combination of characters and
5034		conversion specifications. If the string is empty, the 12-hour format is not
5035		supported in the locale.
5036	<b>era</b>	Define how years are counted and displayed for each era in a locale. The
5037		operand shall consist of semicolon-separated strings. Each string shall be an
5038		era description segment with the format:
5039		<i>direction:offset:start_date:end_date:era_name:era_format</i>
5040		according to the definitions below. There can be as many era description
5041		segments as are necessary to describe the different eras.
5042	<b>Note:</b>	The start of an era might not be the earliest point in the era—it may be the
5043		latest. For example, the Christian era BC starts on the day before January 1,
5044		AD 1, and increases with earlier time.
5045	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate
5046		that years closer to the <i>start_date</i> have lower numbers than those
5047		closer to the <i>end_date</i> . The '-' character shall indicate that
5048		years closer to the <i>start_date</i> have higher numbers than those
5049		closer to the <i>end_date</i> .
5050	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era,
5051		corresponding to the %EY conversion specification.
5052	<i>start_date</i>	A date in the form yyyy/mm/dd, where yyyy, mm, and dd are the
5053		year, month, and day numbers respectively of the start of the
5054		era. Years prior to AD 1 shall be represented as negative
5055		numbers.
5056	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
5057		or one of the two special values "-*" or "+*". The value "-*" shall
5058		indicate that the ending date is the beginning of time. The
5059		value "+*" shall indicate that the ending date is the end of time.
5060	<i>era_name</i>	A string representing the name of the era, corresponding to the
5061		%EC conversion specification.
5062	<i>era_format</i>	A string for formatting the year in the era, corresponding to the
5063		%EY conversion specification.
5064	<b>era_d_fmt</b>	Define the format of the date in alternative era notation, corresponding to the
5065		%Ex conversion specification.
5066	<b>era_t_fmt</b>	Define the locale's appropriate alternative time format, corresponding to the
5067		%EX conversion specification.
5068	<b>era_d_t_fmt</b>	Define the locale's appropriate alternative date and time format,
5069		corresponding to the %Ec conversion specification.
5070	<b>alt_digits</b>	Define alternative symbols for digits, corresponding to the %O modified
5071		conversion specification. The operand shall consist of semicolon-separated
5072		strings, each surrounded by double-quotes. The first string shall be the
5073		alternative symbol corresponding with zero, the second string the symbol

5074 corresponding with one, and so on. Up to 100 alternative symbol strings can  
 5075 be specified. The %O modifier shall indicate that the string corresponding to  
 5076 the value specified via the conversion specification shall be used instead of the  
 5077 value.

### 5078 7.3.5.2 LC\_TIME C-Language Access

5079 XSI This section describes extensions to access information in the *LC\_TIME* category using the  
 5080 *nl\_langinfo()* function. This functionality is dependent on support of the XSI extension (and the  
 5081 rest of this section is not further shaded for this option).

5082 The following constants used to identify items of *langinfo* data can be used as arguments to the  
 5083 *nl\_langinfo()* function to access information in the *LC\_TIME* category. These constants are  
 5084 defined in the **<langinfo.h>** header.

5085 ABDAY\_x The abbreviated weekday names (for example, Sun), where *x* is a number from  
 5086 1 to 7.

5087 DAY\_x The full weekday names (for example, Sunday), where *x* is a number from 1 to  
 5088 7.

5089 ABMON\_x The abbreviated month names (for example, Jan), where *x* is a number from 1  
 5090 to 12.

5091 MON\_x The full month names (for example, January), where *x* is a number from 1 to  
 5092 12.

5093 D\_T\_FMT The appropriate date and time representation.

5094 D\_FMT The appropriate date representation.

5095 T\_FMT The appropriate time representation.

5096 AM\_STR The appropriate ante-meridiem affix.

5097 PM\_STR The appropriate post-meridiem affix.

5098 T\_FMT\_AMPM The appropriate time representation in the 12-hour clock format with  
 5099 AM\_STR and PM\_STR.

5100 ERA The era description segments, which describe how years are counted and  
 5101 displayed for each era in a locale. Each era description segment shall have the  
 5102 format:

5103 *direction:offset:start\_date:end\_date:era\_name:era\_format*

5104 according to the definitions below. There can be as many era description  
 5105 segments as are necessary to describe the different eras. Era description  
 5106 segments are separated by semicolons.

5107 *direction* Either a '+' or a '-' character. The '+' character shall indicate  
 5108 that years closer to the *start\_date* have lower numbers than those  
 5109 closer to the *end\_date*. The '-' character shall indicate that  
 5110 years closer to the *start\_date* have higher numbers than those  
 5111 closer to the *end\_date*.

5112 *offset* The number of the year closest to the *start\_date* in the era.

5113 *start\_date* A date in the form *yyyy/mm/dd*, where *yyyy*, *mm*, and *dd* are the  
 5114 year, month, and day numbers respectively of the start of the  
 5115 era. Years prior to AD 1 shall be represented as negative

5116		numbers.
5117	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
5118		or one of the two special values <i>"-"</i> or <i>"+"</i> . The value <i>"-"</i>
5119		shall indicate that the ending date is the beginning of time. The
5120		value <i>"+"</i> shall indicate that the ending date is the end of time.
5121	<i>era_name</i>	The era, corresponding to the <i>%EC</i> conversion specification.
5122	<i>era_format</i>	The format of the year in the era, corresponding to the <i>%EY</i>
5123		conversion specification.
5124	ERA_D_FMT	The era date format.
5125	ERA_T_FMT	The locale's appropriate alternative time format, corresponding to the <i>%EX</i>
5126		conversion specification.
5127	ERA_D_T_FMT	The locale's appropriate alternative date and time format, corresponding to
5128		the <i>%Ec</i> conversion specification.
5129	ALT_DIGITS	The alternative symbols for digits, corresponding to the <i>%O</i> conversion
5130		specification modifier. The value consists of semicolon-separated symbols.
5131		The first is the alternative symbol corresponding to zero, the second is the
5132		symbol corresponding to one, and so on. Up to 100 alternative symbols may
5133		be specified.

### 5134 7.3.5.3 LC\_TIME Category in the POSIX Locale

5135 The *LC\_TIME* category definition of the POSIX locale follows; the code listing depicts the  
5136 *localedef* input; the table represents the same information with the addition of *localedef* keywords,  
5137 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*  
5138 XSI functions, and *nl\_langinfo()* constants.

```

5139 LC_TIME
5140 # This is the POSIX locale definition for
5141 # the LC_TIME category.
5142 #
5143 # Abbreviated weekday names (%a)
5144 abday      "<S><u><n>" ; "<M><o><n>" ; "<T><u><e>" ; "<W><e><d>" ; \
5145            "<T><h><u>" ; "<F><r><i>" ; "<S><a><t>"
5146 #
5147 # Full weekday names (%A)
5148 day        "<S><u><n><d><a><y>" ; "<M><o><n><d><a><y>" ; \
5149            "<T><u><e><s><d><a><y>" ; "<W><e><d><n><e><s><d><a><y>" ; \
5150            "<T><h><u><r><s><d><a><y>" ; "<F><r><i><d><a><y>" ; \
5151            "<S><a><t><u><r><d><a><y>"
5152 #
5153 # Abbreviated month names (%b)
5154 abmon      "<J><a><n>" ; "<F><e><b>" ; "<M><a><r>" ; \
5155            "<A><p><r>" ; "<M><a><y>" ; "<J><u><n>" ; \
5156            "<J><u><l>" ; "<A><u><g>" ; "<S><e><p>" ; \
5157            "<O><c><t>" ; "<N><o><v>" ; "<D><e><c>"
5158 #
5159 # Full month names (%B)
5160 mon        "<J><a><n><u><a><r><y>" ; "<F><e><b><r><u><a><r><y>" ; \
5161            "<M><a><r><c><h>" ; "<A><p><r><i><l>" ; \
5162            "<M><a><y>" ; "<J><u><n><e>" ; \

```

```

5163         "<J><u><l><y>" ; "<A><u><g><u><s><t>" ; \
5164         "<S><e><p><t><e><m><b><e><r>" ; "<O><c><t><o><b><e><r>" ; \
5165         "<N><o><v><e><m><b><e><r>" ; "<D><e><c><e><m><b><e><r>"
5166     #
5167     # Equivalent of AM/PM (%p)          "AM"; "PM"
5168     am_pm          "<A><M>" ; "<P><M>"
5169     #
5170     # Appropriate date and time representation (%c)
5171     #      "%a %b %e %H:%M:%S %Y"
5172     d_t_fmt        "<percent-sign><a><space><percent-sign><b>\
5173     <space><percent-sign><e><space><percent-sign><H>\
5174     <colon><percent-sign><M><colon><percent-sign><S>\
5175     <space><percent-sign><Y>"
5176     #
5177     # Appropriate date representation (%x)  "%m/%d/%y"
5178     d_fmt          "<percent-sign><m><slash><percent-sign><d>\
5179     <slash><percent-sign><y>"
5180     #
5181     # Appropriate time representation (%X)  "%H:%M:%S"
5182     t_fmt          "<percent-sign><H><colon><percent-sign><M>\
5183     <colon><percent-sign><S>"
5184     #
5185     # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5186     t_fmt_ampm     "<percent-sign><I><colon><percent-sign><M><colon>\
5187     <percent-sign><S><space><percent_sign><p>"
5188     #
5189     END LC_TIME

```

5190

5191

5192

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
d_fmt	D_FMT	%x	"%m/%d/%y"
t_fmt	T_FMT	%X	"%H:%M:%S"
am_pm	AM_STR	%p	"AM"
am_pm	PM_STR	%p	"PM"
t_fmt_ampm	T_FMT_AMP	%r	"%I:%M:%S %p"
day	DAY_1	%A	"Sunday"
day	DAY_2	%A	"Monday"
day	DAY_3	%A	"Tuesday"
day	DAY_4	%A	"Wednesday"
day	DAY_5	%A	"Thursday"
day	DAY_6	%A	"Friday"
day	DAY_7	%A	"Saturday"
abday	ABDAY_1	%a	"Sun"
abday	ABDAY_2	%a	"Mon"
abday	ABDAY_3	%a	"Tue"
abday	ABDAY_4	%a	"Wed"
abday	ABDAY_5	%a	"Thu"

5211  
5212  
5213  
5214  
5215  
5216  
5217  
5218  
5219  
5220  
5221  
5222  
5223  
5224  
5225  
5226  
5227  
5228  
5229  
5230  
5231  
5232  
5233  
5234  
5235  
5236  
5237  
5238  
5239  
5240  
5241  
5242  
5243  
5244

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
abday	ABDAY_6	%a	"Fri "
abday	ABDAY_7	%a	"Sat "
mon	MON_1	%B	"January"
mon	MON_2	%B	"February"
mon	MON_3	%B	"March"
mon	MON_4	%B	"April "
mon	MON_5	%B	"May"
mon	MON_6	%B	"June"
mon	MON_7	%B	"July"
mon	MON_8	%B	"August "
mon	MON_9	%B	"September "
mon	MON_10	%B	"October "
mon	MON_11	%B	"November "
mon	MON_12	%B	"December "
abmon	ABMON_1	%b	"Jan "
abmon	ABMON_2	%b	"Feb "
abmon	ABMON_3	%b	"Mar "
abmon	ABMON_4	%b	"Apr "
abmon	ABMON_5	%b	"May"
abmon	ABMON_6	%b	"Jun "
abmon	ABMON_7	%b	"Jul "
abmon	ABMON_8	%b	"Aug "
abmon	ABMON_9	%b	"Sep "
abmon	ABMON_10	%b	"Oct "
abmon	ABMON_11	%b	"Nov "
abmon	ABMON_12	%b	"Dec "
era	ERA	%EC, %Ey, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

5245 XSI

In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5246

The entry N/A indicates the value is not available in the POSIX locale.

### 5247 7.3.6 LC\_MESSAGES

5248

The *LC\_MESSAGES* category shall define the format and values used by various utilities for affirmative and negative responses. This information is available through the *nl\_langinfo()* function.

5249 XSI

5250

5251 XSI

The message catalog used by the standard utilities and selected by the *catopen()* function shall be determined by the setting of *NLSPATH*; see Chapter 8 (on page 161). The *LC\_MESSAGES* category can be specified as part of an *NLSPATH* substitution field.

5252

5253

5254

The following keywords shall be recognized as part of the locale definition file.

5255

**copy**

Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

5256

5257

**Note:** This is a *localedef* keyword, unavailable through *nl\_langinfo()*.

5258           **yesexpr**       The operand consists of an extended regular expression (see Section 9.4 (on page

5259                               175)) that describes the acceptable affirmative response to a question expecting an

5260                               affirmative or negative response.

5261           **noexpr**        The operand consists of an extended regular expression that describes the

5262                               acceptable negative response to a question expecting an affirmative or negative

5263                               response.

5264   7.3.6.1   **LC\_MESSAGES Category in the POSIX Locale**

5265           The format and values for affirmative and negative responses of the POSIX locale follow; the

5266   XSI       code listing depicting the *localedef* input, the table representing the same information with the

5267           addition of *nl\_langinfo()* constants.

```
5268   LC_MESSAGES
5269   # This is the POSIX locale definition for
5270   # the LC_MESSAGES category.
5271   #
5272   yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5273   #
5274   noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5275   #
5276   END LC_MESSAGES
```

5277		<b>localedef Keyword</b>	<b>langinfo Constant</b>	<b>POSIX Locale Value</b>
5278		<b>yesexpr</b>	YESEXPR	"^[yY]"
5279		<b>noexpr</b>	NOEXPR	"^[nN]"

5280   XSI       In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5281   **7.4    Locale Definition Grammar**

5282           The grammar and lexical conventions in this section shall together describe the syntax for the

5283           locale definition source. The general conventions for this style of grammar are described in the

5284           Shell and Utilities volume of IEEE Std 1003.1-2001, Section 1.10, Grammar Conventions. The

5285           grammar shall take precedence over the text in this chapter.

5286   7.4.1   **Locale Lexical Conventions**

5287           The lexical conventions for the locale definition grammar are described in this section.

5288           The following tokens shall be processed (in addition to those string constants shown in the

5289           grammar):

5290	<b>LOC_NAME</b>	A string of characters representing the name of a locale.
5291	<b>CHAR</b>	Any single character.
5292	<b>NUMBER</b>	A decimal number, represented by one or more decimal digits.
5293	<b>COLLSYMBOL</b>	A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a <b>COLLELEMENT</b> symbol.
5294		
5295		
5296	<b>COLLELEMENT</b>	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a <b>COLLSYMBOL</b> symbol.
5297		

5298	<b>CHARCLASS</b>	A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.
5299		
5300		
5301		
5302	<b>CHARSYMBOL</b>	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
5303		
5304	<b>OCTAL_CHAR</b>	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a backslash) followed by two or more octal digits.
5305		
5306		
5307		
5308	<b>HEX_CHAR</b>	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant <i>x</i> and two or more hexadecimal digits.
5309		
5310		
5311		
5312	<b>DECIMAL_CHAR</b>	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.
5313		
5314		
5315		
5316	<b>ELLIPSIS</b>	The string " . . . ".
5317	<b>EXTENDED_REG_EXP</b>	An extended regular expression as defined in the grammar in Section 9.5 (on page 179).
5318		
5319	<b>EOL</b>	The line termination character <newline>.

## 5320 7.4.2 Locale Grammar

5321	This section presents the grammar for the locale definition.	
5322	%token	LOC_NAME
5323	%token	CHAR
5324	%token	NUMBER
5325	%token	COLLSYMBOL COLLELEMENT
5326	%token	CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5327	%token	ELLIPSIS
5328	%token	EXTENDED_REG_EXP
5329	%token	EOL
5330	%start	locale_definition
5331	%%	
5332	locale_definition	: global_statements locale_categories
5333		locale_categories
5334		;
5335	global_statements	: global_statements symbol_redefine
5336		symbol_redefine
5337		;
5338	symbol_redefine	: 'escape_char' CHAR EOL
5339		'comment_char' CHAR EOL
5340		;



```

5341     locale_categories      : locale_categories locale_category
5342                             | locale_category
5343                             ;
5344     locale_category         : lc_ctype | lc_collate | lc_messages
5345                             | lc_monetary | lc_numeric | lc_time
5346                             ;
5347     /* The following grammar rules are common to all categories */
5348     char_list                : char_list char_symbol
5349                             | char_symbol
5350                             ;
5351     char_symbol              : CHAR | CHARSYMBOL
5352                             | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5353                             ;
5354     elem_list                : elem_list char_symbol
5355                             | elem_list COLLSYMBOL
5356                             | elem_list COLLELEMENT
5357                             | char_symbol
5358                             | COLLSYMBOL
5359                             | COLLELEMENT
5360                             ;
5361     symb_list                : symb_list COLLSYMBOL
5362                             | COLLSYMBOL
5363                             ;
5364     locale_name              : LOC_NAME
5365                             | ' ' LOC_NAME ' '
5366                             ;
5367     /* The following is the LC_CTYPE category grammar */
5368     lc_ctype                  : ctype_hdr ctype_keywords          ctype_tlr
5369                             | ctype_hdr 'copy' locale_name EOL ctype_tlr
5370                             ;
5371     ctype_hdr                : 'LC_CTYPE' EOL
5372                             ;
5373     ctype_keywords           : ctype_keywords ctype_keyword
5374                             | ctype_keyword
5375                             ;
5376     ctype_keyword            : charclass_keyword charclass_list EOL
5377                             | charconv_keyword charconv_list EOL
5378                             | 'charclass' charclass_namelist EOL
5379                             ;
5380     charclass_namelist       : charclass_namelist ';' CHARCLASS
5381                             | CHARCLASS
5382                             ;
5383     charclass_keyword        : 'upper' | 'lower' | 'alpha' | 'digit'
5384                             | 'punct' | 'xdigit' | 'space' | 'print'
5385                             | 'graph' | 'blank' | 'cntrl' | 'alnum'

```

```

5386          | CHARCLASS
5387          ;

5388      charclass_list      : charclass_list ';' char_symbol
5389          | charclass_list ';' ELLIPSIS ';' char_symbol
5390          | char_symbol
5391          ;

5392      charconv_keyword    : 'toupper'
5393          | 'tolower'
5394          ;

5395      charconv_list       : charconv_list ';' charconv_entry
5396          | charconv_entry
5397          ;

5398      charconv_entry      : '(' char_symbol ',' char_symbol ')'
5399          ;

5400      ctype_tlr           : 'END' 'LC_CTYPE' EOL
5401          ;

5402      /* The following is the LC_COLLATE category grammar */

5403      lc_collate           : collate_hdr collate_keywords      collate_tlr
5404          | collate_hdr 'copy' locale_name EOL collate_tlr
5405          ;

5406      collate_hdr          : 'LC_COLLATE' EOL
5407          ;

5408      collate_keywords     :          order_statements
5409          | opt_statements order_statements
5410          ;

5411      opt_statements       : opt_statements collating_symbols
5412          | opt_statements collating_elements
5413          | collating_symbols
5414          | collating_elements
5415          ;

5416      collating_symbols    : 'collating-symbol' COLLSYMBOL EOL
5417          ;

5418      collating_elements   : 'collating-element' COLLELEMENT
5419          | 'from' '"' elem_list '"' EOL
5420          ;

5421      order_statements     : order_start collation_order order_end
5422          ;

5423      order_start          : 'order_start' EOL
5424          | 'order_start' order_opts EOL
5425          ;

5426      order_opts           : order_opts ';' order_opt
5427          | order_opt
5428          ;

```

```

5429      order_opt      : order_opt ',' opt_word
5430                        | opt_word
5431                        ;
5432      opt_word         : 'forward' | 'backward' | 'position'
5433                        ;
5434      collation_order  : collation_order collation_entry
5435                        | collation_entry
5436                        ;
5437      collation_entry  : COLLSYMBOL EOL
5438                        | collation_element weight_list EOL
5439                        | collation_element EOL
5440                        ;
5441      collation_element: char_symbol
5442                        | COLLELEMENT
5443                        | ELLIPSIS
5444                        | 'UNDEFINED'
5445                        ;
5446      weight_list      : weight_list ';' weight_symbol
5447                        | weight_list ';'
5448                        | weight_symbol
5449                        ;
5450      weight_symbol    : /* empty */
5451                        | char_symbol
5452                        | COLLSYMBOL
5453                        | ''' elem_list '''
5454                        | ''' symb_list '''
5455                        | ELLIPSIS
5456                        | 'IGNORE'
5457                        ;
5458      order_end        : 'order_end' EOL
5459                        ;
5460      collate_tlr       : 'END' 'LC_COLLATE' EOL
5461                        ;
5462      /* The following is the LC_MESSAGES category grammar */
5463      lc_messages       : messages_hdr messages_keywords messages_tlr
5464                        | messages_hdr 'copy' locale_name EOL messages_tlr
5465                        ;
5466      messages_hdr      : 'LC_MESSAGES' EOL
5467                        ;
5468      messages_keywords : messages_keywords messages_keyword
5469                        | messages_keyword
5470                        ;
5471      messages_keyword  : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5472                        | 'noexpr' ''' EXTENDED_REG_EXP ''' EOL
5473                        ;

```

```

5474     messages_tlr           : 'END' 'LC_MESSAGES' EOL
5475                               ;

5476     /* The following is the LC_MONETARY category grammar */

5477     lc_monetary              : monetary_hdr monetary_keywords      monetary_tlr
5478                               | monetary_hdr 'copy' locale_name EOL monetary_tlr
5479                               ;

5480     monetary_hdr             : 'LC_MONETARY' EOL
5481                               ;

5482     monetary_keywords        : monetary_keywords monetary_keyword
5483                               | monetary_keyword
5484                               ;

5485     monetary_keyword         : mon_keyword_string mon_string EOL
5486                               | mon_keyword_char NUMBER EOL
5487                               | mon_keyword_char '-1' EOL
5488                               | mon_keyword_grouping mon_group_list EOL
5489                               ;

5490     mon_keyword_string       : 'int_curr_symbol' | 'currency_symbol'
5491                               | 'mon_decimal_point' | 'mon_thousands_sep'
5492                               | 'positive_sign' | 'negative_sign'
5493                               ;

5494     mon_string               : '"' char_list '"'
5495                               | '""'
5496                               ;

5497     mon_keyword_char         : 'int_frac_digits' | 'frac_digits'
5498                               | 'p_cs_precedes' | 'p_sep_by_space'
5499                               | 'n_cs_precedes' | 'n_sep_by_space'
5500                               | 'p_sign_posn' | 'n_sign_posn'
5501                               | 'int_p_cs_precedes' | 'int_p_sep_by_space' 1
5502                               | 'int_n_cs_precedes' | 'int_n_sep_by_space' 1
5503                               | 'int_p_sign_posn' | 'int_n_sign_posn' 1
5504                               ;

5505     mon_keyword_grouping     : 'mon_grouping'
5506                               ;

5507     mon_group_list           : NUMBER
5508                               | mon_group_list ';' NUMBER
5509                               ;

5510     monetary_tlr            : 'END' 'LC_MONETARY' EOL
5511                               ;

5512     /* The following is the LC_NUMERIC category grammar */

5513     lc_numeric               : numeric_hdr numeric_keywords      numeric_tlr
5514                               | numeric_hdr 'copy' locale_name EOL numeric_tlr
5515                               ;

5516     numeric_hdr              : 'LC_NUMERIC' EOL
5517                               ;

```

```

5518     numeric_keywords      : numeric_keywords numeric_keyword
5519                             | numeric_keyword
5520                             ;
5521     numeric_keyword        : num_keyword_string num_string EOL
5522                             | num_keyword_grouping num_group_list EOL
5523                             ;
5524     num_keyword_string     : 'decimal_point'
5525                             | 'thousands_sep'
5526                             ;
5527     num_string             : ''' char_list '''
5528                             | '""'
5529                             ;
5530     num_keyword_grouping   : 'grouping'
5531                             ;
5532     num_group_list         : NUMBER
5533                             | num_group_list ';' NUMBER
5534                             ;
5535     numeric_tlr            : 'END' 'LC_NUMERIC' EOL
5536                             ;
5537     /* The following is the LC_TIME category grammar */
5538     lc_time                 : time_hdr time_keywords          time_tlr
5539                             | time_hdr 'copy' locale_name EOL time_tlr
5540                             ;
5541     time_hdr               : 'LC_TIME' EOL
5542                             ;
5543     time_keywords          : time_keywords time_keyword
5544                             | time_keyword
5545                             ;
5546     time_keyword           : time_keyword_name time_list EOL
5547                             | time_keyword_fmt time_string EOL
5548                             | time_keyword_opt time_list EOL
5549                             ;
5550     time_keyword_name      : 'abday' | 'day' | 'abmon' | 'mon'
5551                             ;
5552     time_keyword_fmt       : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5553                             | 'am_pm' | 't_fmt_ampm'
5554                             ;
5555     time_keyword_opt       : 'era' | 'era_d_fmt' | 'era_t_fmt'
5556                             | 'era_d_t_fmt' | 'alt_digits'
5557                             ;
5558     time_list              : time_list ';' time_string
5559                             | time_string
5560                             ;

```

```
5561      time_string      : ' ' char_list ' '
5562                          ;
5563      time_tlr           : 'END' 'LC_TIME' EOL
5564                          ;
```

# Environment Variables

5565

## 8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of IEEE Std 1003.1-2001 for information on environment variable usage.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name=value*; *names* shall not contain the character '='. For values to be portable across systems conforming to IEEE Std 1003.1-2001, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in a process' environment has the same *name*, the consequences are undefined.

Environment variable names used by the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 consist solely of uppercase letters, digits, and the '\_' (underscore) from the characters defined in Table 6-1 (on page 113) and do not begin with a digit. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

**Note:** Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG\_MAX} bytes.

Other *name=value* pairs may be placed in the environment by, for example, calling any of the *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of IEEE Std 1003.1-2001.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

5601	ARFLAGS	IFS	MAILPATH	PS1
5602	CC	LANG	MAILRC	PS2
5603	CDPATH	LC_ALL	MAKEFLAGS	PS3
5604	CFLAGS	LC_COLLATE	MAKESHELL	PS4
5605	CHARSET	LC_CTYPE	MANPATH	PWD
5606	COLUMNS	LC_MESSAGES	MBOX	RANDOM
5607	DATMSK	LC_MONETARY	MORE	SECONDS
5608	DEAD	LC_NUMERIC	MSGVERB	SHELL
5609	EDITOR	LC_TIME	NLSPATH	TERM
5610	ENV	LDFLAGS	NPROC	TERMCAP
5611	EXINIT	LEX	OLDPWD	TERMINFO
5612	FC	LFLAGS	OPTARG	TMPDIR
5613	FCEDIT	LINENO	OPTERR	TZ
5614	FFLAGS	LINES	OPTIND	USER
5615	GET	LISTER	PAGER	VISUAL
5616	GFLAGS	LOGNAME	PATH	YACC
5617	HISTFILE	LPDEST	PPID	YFLAGS
5618	HISTORY	MAIL	PRINTER	
5619	HISTSIZE	MAILCHECK	PROCLANG	
5620	HOME	MAILER	PROJECTDIR	

If the variables in the following two sections are present in the environment during the execution of an application or utility, they shall be given the meaning described below. Some are placed into the environment by the implementation at the time the user logs in; all can be added or changed by the user or any ancestor of the current process. The implementation adds or changes environment variables named in IEEE Std 1003.1-2001 only as specified in IEEE Std 1003.1-2001. If they are defined in the application's environment, the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 and the functions in the System Interfaces volume of IEEE Std 1003.1-2001 assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See *getenv()* and the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.12, Shell Execution Environment for methods of accessing these variables.

## 8.2 Internationalization Variables

This section describes environment variables that are relevant to the operation of internationalized interfaces described in IEEE Std 1003.1-2001.

Users may use the following environment variables to announce specific localization requirements to applications. Applications can retrieve this information using the *setlocale()* function to initialize the correct behavior of the internationalized interfaces. The descriptions of the internationalization environment variables describe the resulting behavior only when the application locale is initialized in this way. The use of the internationalization variables by utilities described in the Shell and Utilities volume of IEEE Std 1003.1-2001 is described in the ENVIRONMENT VARIABLES section for those utilities in addition to the global effects described in this section.

**LANG** This variable shall determine the locale category for native language, local customs, and coded character set in the absence of the *LC\_ALL* and other *LC\_\** (*LC\_COLLATE*, *LC\_CTYPE*, *LC\_MESSAGES*, *LC\_MONETARY*, *LC\_NUMERIC*, *LC\_TIME*) environment variables. This can be used by applications to determine the language to use for error messages and instructions, collating sequences, date formats, and so on.



5649	<b>LC_ALL</b>	This variable shall determine the values for all locale categories. The value of the <b>LC_ALL</b> environment variable has precedence over any of the other environment variables starting with <b>LC_</b> ( <b>LC_COLLATE</b> , <b>LC_CTYPE</b> , <b>LC_MESSAGES</b> , <b>LC_MONETARY</b> , <b>LC_NUMERIC</b> , <b>LC_TIME</b> ) and the <b>LANG</b> environment variable.
5650		
5651		
5652		
5653		
5654	<b>LC_COLLATE</b>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <b>strcoll()</b> and <b>strxfrm()</b> functions. Additional semantics of this variable, if any, are implementation-defined.
5655		
5656		
5657		
5658		
5659	<b>LC_CTYPE</b>	This variable shall determine the locale category for character handling functions, such as <b>tolower()</b> , <b>toupper()</b> , and <b>isalpha()</b> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5660		
5661		
5662		
5663		
5664		
5665		
5666	<b>LC_MESSAGES</b>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <b>catopen()</b>
5667		
5668	XSI	function in determining the message catalog. Additional semantics of this
5669		variable, if any, are implementation-defined. The language and cultural
5670		conventions of diagnostic and informative messages whose format is
5671		unspecified by IEEE Std 1003.1-2001 should be affected by the setting of
5672		<b>LC_MESSAGES</b> .
5673		
5674	<b>LC_MONETARY</b>	This variable shall determine the locale category for monetary-related numeric
5675		formatting information. Additional semantics of this variable, if any, are
5676		implementation-defined.
5677	<b>LC_NUMERIC</b>	This variable shall determine the locale category for numeric formatting (for
5678		example, thousands separator and radix character) information in various
5679		utilities as well as the formatted I/O operations in <b>printf()</b> and <b>scanf()</b> and the
5680		string conversion functions in <b>strtod()</b> . Additional semantics of this variable,
5681		if any, are implementation-defined.
5682	<b>LC_TIME</b>	This variable shall determine the locale category for date and time formatting
5683		information. It affects the behavior of the time functions in <b>strftime()</b> .
5684		Additional semantics of this variable, if any, are implementation-defined.
5685	XSI	<b>NLSPATH</b> This variable shall contain a sequence of templates that the <b>catopen()</b> function
5686		uses when attempting to locate message catalogs. Each template consists of an
5687		optional prefix, one or more conversion specifications, a filename, and an
5688		optional suffix.
5689		For example:
5690		<b>NLSPATH="/system/nlslib/%N.cat"</b>
5691		defines that <b>catopen()</b> should look for all message catalogs in the directory
5692		<b>/system/nlslib</b> , where the catalog name should be constructed from the <i>name</i>
5693		parameter passed to <b>catopen()</b> ( <b>%N</b> ), with the suffix <b>.cat</b> .
5694		Conversion specifications consist of a <b>'%'</b> symbol, followed by a single-letter
5695		keyword. The following keywords are currently defined:

5696	%N	The value of the <i>name</i> parameter passed to <i>catopen()</i> .
5697	%L	The value of the <i>LC_MESSAGES</i> category.
5698	%l	The <i>language</i> element from the <i>LC_MESSAGES</i> category.
5699	%t	The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
5700	%c	The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
5701	%%	A single ' % ' character.
5702	An empty string is substituted if the specified value is not currently defined.	
5703	The separators underscore ( ' _ ' ) and period ( ' . ' ) are not included in the %t	
5704	and %c conversion specifications.	
5705	Templates defined in <i>NLSPATH</i> are separated by colons ( ' : ' ). A leading or	
5706	two adjacent colons " : : " is equivalent to specifying %N. For example:	
5707	<i>NLSPATH</i> =" : %N.cat : /nlslib/%L/%N.cat "	
5708	indicates to <i>catopen()</i> that it should look for the requested message catalog in	
5709	<i>name</i> , <i>name.cat</i> , and <i>/nlslib/category/name.cat</i> , where <i>category</i> is the value of the	
5710	<i>LC_MESSAGES</i> category of the current locale.	
5711	Users should not set the <i>NLSPATH</i> variable unless they have a specific reason	
5712	to override the default system path. Setting <i>NLSPATH</i> to override the default	
5713	system path produces undefined results in the standard utilities and in	
5714	applications with appropriate privileges.	
5715	XSI	The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5716		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of
5717		internationalized applications. The standard utilities shall make use of these environment
5718		variables as described in this section and the individual ENVIRONMENT VARIABLES sections
5719		for the utilities. If these variables specify locale categories that are not based upon the same
5720		underlying codeset, the results are unspecified.
5721	The values of locale categories shall be determined by a precedence order; the first condition met	
5722	below determines the value:	
5723	1. If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall be	
5724	used.	
5725	2. If the <i>LC_*</i> environment variable ( <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,	
5726	<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> ) is defined and is not null, the value of the	
5727	environment variable shall be used to initialize the category that corresponds to the	
5728	environment variable.	
5729	3. If the <i>LANG</i> environment variable is defined and is not null, the value of the <i>LANG</i>	
5730	environment variable shall be used.	
5731	4. If the <i>LANG</i> environment variable is not set or is set to the empty string, the	
5732	implementation-defined default locale shall be used.	
5733	If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities	
5734	behave in accordance with the rules in Section 7.2 (on page 124) for the associated category.	
5735	If the locale value begins with a slash, it shall be interpreted as the pathname of a file that was	
5736	created in the output format used by the <i>localedf</i> utility; see OUTPUT FILES under <i>localedf</i> .	
5737	Referencing such a pathname shall result in that locale being used for the indicated category.	

5738	XSI	If the locale value has the form:
5739		<code>language[_territory][.codeset]</code>
5740		it refers to an implementation-provided locale, where settings of language, territory, and codeset
5741		are implementation-defined.
5742		<code>LC_COLLATE</code> , <code>LC_CTYPE</code> , <code>LC_MESSAGES</code> , <code>LC_MONETARY</code> , <code>LC_NUMERIC</code> , and <code>LC_TIME</code> are
5743		defined to accept an additional field <code>@modifier</code> , which allows the user to select a specific instance
5744		of localization data within a single category (for example, for selecting the dictionary as opposed
5745		to the character ordering of data). The syntax for these environment variables is thus defined as:
5746		<code>[language[_territory][.codeset][@modifier]]</code>
5747		For example, if a user wanted to interact with the system in French, but required to sort German
5748		text files, <code>LANG</code> and <code>LC_COLLATE</code> could be defined as:
5749		<code>LANG=Fr_FR</code>
5750		<code>LC_COLLATE=De_DE</code>
5751		This could be extended to select dictionary collation (say) by use of the <code>@modifier</code> field; for
5752		example:
5753		<code>LC_COLLATE=De_DE@dict</code>
5754		
5755		An implementation may support other formats.
5756		If the locale value is not recognized by the implementation, the behavior is unspecified.
5757		At runtime, these values are bound to a program's locale by calling the <code>setlocale()</code> function.
5758		Additional criteria for determining a valid locale name are implementation-defined.

### 5759 8.3 Other Environment Variables

5760	<code>COLUMNS</code>	This variable shall represent a decimal integer >0 used to indicate the user's preferred width in column positions for the terminal screen or window; see Section 3.103 (on page 49). If this variable is unset or null, the implementation determines the number of columns, appropriate for the terminal or window, in an unspecified manner. When <code>COLUMNS</code> is set, any terminal-width information implied by <code>TERM</code> is overridden. Users and conforming applications should not set <code>COLUMNS</code> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
5761		
5762		
5763		
5764		
5765		
5766		
5767		
5768		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
5769		
5770		
5771	XSI	<code>DATETIME</code> Indicates the pathname of the template file used by <code>getdate()</code> .
5772	<code>HOME</code>	The system shall initialize this variable at the time of login to be a pathname of the user's home directory. See <code>&lt;pwd.h&gt;</code> .
5773		
5774	<code>LINES</code>	This variable shall represent a decimal integer >0 used to indicate the user's preferred number of lines on a page or the vertical screen or window size in lines. A line in this case is a vertical measure large enough to hold the tallest character in the character set being displayed. If this variable is unset or null, the implementation determines the number of lines, appropriate for the
5775		
5776		
5777		
5778		

5779		terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When <i>LINES</i> is set, any terminal-height information implied by <i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
5780		
5781		
5782		
5783		
5784		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
5785		
5786		
5787	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's login name. See <pwd.h>. For a value of <i>LOGNAME</i> to be portable across implementations of IEEE Std 1003.1-2001, the value should be composed of characters from the portable filename character set.
5788		
5789		
5790		
5791	XSI	<i>MSGVERB</i> Describes which message components shall be used in writing messages by <i>fmtmsg()</i> .
5792		
5793	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a filename. The prefixes shall be separated by a colon (':'). When a non-zero-length prefix is applied to this filename, a slash shall be inserted between the prefix and the filename. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent colons (": :"), as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. A strictly conforming application shall use an actual pathname (such as .) to represent the current working directory in <i>PATH</i> . The list shall be searched from beginning to end, applying the filename to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the pathname being sought contains a slash, the search through the path prefixes shall not be performed. If the pathname begins with a slash, the specified path is resolved (see Section 4.11 (on page 100)). If <i>PATH</i> is unset or is set to null, the path search is implementation-defined.
5794		
5795		
5796		
5797		
5798		
5799		
5800		
5801		
5802		
5803		
5804		
5805		
5806		
5807		
5808		
5809	<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. It shall not contain any filename components of dot or dot-dot. The value is set by the <i>cd</i> utility.
5810		
5811		
5812	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in the Shell and Utilities volume of IEEE Std 1003.1-2001, Chapter 2, Shell Command Language, utilities may behave differently from those described in IEEE Std 1003.1-2001.
5813		
5814		
5815		
5816		
5817	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.
5818		
5819	<i>TERM</i>	This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
5820		
5821		
5822		
5823	<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>localtime()</i> , <i>strftime()</i> , <i>mktime()</i> , <i>ctime_r()</i> , and <i>localtime_r()</i> functions, and by various utilities, to override the default timezone. The value of <i>TZ</i> has one of the two
5824		
5825	TSF	
5826		

5827 forms (spaces inserted for clarity):

5828 `:characters`

5829 or:

5830 `std offset dst offset, rule`

5831 If *TZ* is of the first format (that is, if the first character is a colon), the  
5832 characters following the colon are handled in an implementation-defined  
5833 manner.

5834 The expanded format (for all *TZs* whose value does not have a colon as the  
5835 first character) is as follows:

5836 `stdoffset[dst[offset][,start[/time],end[/time]]]`

5837 Where:

5838 *std* and *dst* Indicate no less than three, nor more than {TZNAME\_MAX},  
5839 bytes that are the designation for the standard (*std*) or the  
5840 alternative (*dst*—such as Daylight Savings Time) timezone. Only  
5841 *std* is required; if *dst* is missing, then the alternative time does  
5842 not apply in this locale.

5843 Each of these fields may occur in either of two formats quoted or  
5844 unquoted:

5845 — In the quoted form, the first character shall be the less-than  
5846 ('<') character and the last character shall be the greater-  
5847 than ('>') character. All characters between these quoting  
5848 characters shall be alphanumeric characters from the  
5849 portable character set in the current locale, the plus-sign  
5850 ('+') character, or the minus-sign ('-') character. The *std*  
5851 and *dst* fields in this case shall not include the quoting  
5852 characters.

5853 — In the unquoted form, all characters in these fields shall be  
5854 alphabetic characters from the portable character set in the  
5855 current locale.

5856 The interpretation of these fields is unspecified if either field is  
5857 less than three bytes (except for the case when *dst* is missing),  
5858 more than {TZNAME\_MAX} bytes, or if they contain characters  
5859 other than those specified.

5860 *offset* Indicates the value added to the local time to arrive at  
5861 Coordinated Universal Time. The *offset* has the form:

5862 `hh[:mm[:ss]]`

5863 The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)  
5864 shall be required and may be a single digit. The *offset* following  
5865 *std* shall be required. If no *offset* follows *dst*, the alternative time  
5866 is assumed to be one hour ahead of standard time. One or more  
5867 digits may be used; the value is always interpreted as a decimal  
5868 number. The hour shall be between zero and 24, and the minutes  
5869 (and seconds)—if present—between zero and 59. The result of  
5870 using values outside of this range is unspecified. If preceded by  
5871 a '-', the timezone shall be east of the Prime Meridian;

5872		otherwise, it shall be west (which may be indicated by an
5873		optional preceding '+').
5874	<i>rule</i>	Indicates when to change to and back from the alternative time.
5875		The <i>rule</i> has the form:
5876		<code>date[/time],date[/time]</code>
5877		where the first <i>date</i> describes when the change from standard to
5878		alternative time occurs and the second <i>date</i> describes when the
5879		change back happens. Each <i>time</i> field describes when, in current
5880		local time, the change to the other time is made.
5881		The format of <i>date</i> is one of the following:
5882	<i>Jn</i>	The Julian day $n$ ( $1 \leq n \leq 365$ ). Leap days shall not be
5883		counted. That is, in all years—including leap years—
5884		February 28 is day 59 and March 1 is day 60. It is
5885		impossible to refer explicitly to the occasional February
5886		29.
5887	<i>n</i>	The zero-based Julian day ( $0 \leq n \leq 365$ ). Leap days shall
5888		be counted, and it is possible to refer to February 29.
5889	<i>Mm.n.d</i>	The $d$ 'th day ( $0 \leq d \leq 6$ ) of week $n$ of month $m$ of the
5890		year ( $1 \leq n \leq 5$ , $1 \leq m \leq 12$ , where week 5 means “the
5891		last $d$ day in month $m$ ” which may occur in either the
5892		fourth or the fifth week). Week 1 is the first week in
5893		which the $d$ 'th day occurs. Day zero is Sunday.
5894		The <i>time</i> has the same format as <i>offset</i> except that no leading sign
5895		(‘-’ or ‘+’) is allowed. The default, if <i>time</i> is not given, shall be
5896		02:00:00.

# Regular Expressions

5897

5898 Regular Expressions (REs) provide a mechanism to select specific strings from a set of character  
5899 strings.

5900 Regular expressions are a context-independent syntax that can represent a wide variety of  
5901 character sets and character set orderings, where these character sets are interpreted according  
5902 to the current locale. While many regular expressions can be interpreted differently depending  
5903 on the current locale, many features, such as character class expressions, provide for contextual  
5904 invariance across locales.

5905 The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 171)  
5906 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the  
5907 Extended Regular Expressions (ERE) described in Section 9.4 (on page 175); any exceptions for  
5908 both cases are noted in the descriptions of the specific utilities using regular expressions. Both  
5909 BREs and EREs are supported by the Regular Expression Matching interface in the System  
5910 Interfaces volume of IEEE Std 1003.1-2001 under *regcomp()*, *regex()*, and related functions.

## 5911 9.1 Regular Expression Definitions

5912 For the purposes of this section, the following definitions shall apply:

### 5913 **entire regular expression**

5914 The concatenated set of one or more BREs or EREs that make up the pattern specified for  
5915 string selection.

### 5916 **matched**

5917 A sequence of zero or more characters shall be said to be matched by a BRE or ERE when  
5918 the characters in the sequence correspond to a sequence of characters defined by the  
5919 pattern.

5920 Matching shall be based on the bit pattern used for encoding the character, not on the  
5921 graphic representation of the character. This means that if a character set contains two or  
5922 more encodings for a graphic symbol, or if the strings searched contain text encoded in  
5923 more than one codeset, no attempt is made to search for any other representation of the  
5924 encoded symbol. If that is required, the user can specify equivalence classes containing all  
5925 variations of the desired graphic symbol.

5926 The search for a matching sequence starts at the beginning of a string and stops when the  
5927 first sequence matching the expression is found, where “first” is defined to mean “begins  
5928 earliest in the string”. If the pattern permits a variable number of matching characters and  
5929 thus there is more than one such sequence starting at that point, the longest such sequence  
5930 is matched. For example, the BRE “bb\*” matches the second to fourth characters of the  
5931 string “abbbc”, and the ERE “(wee|week)(knights|night)” matches all ten  
5932 characters of the string “weeknights”.

5933 Consistent with the whole match being the longest of the leftmost matches, each subpattern,  
5934 from left to right, shall match the longest possible string. For this purpose, a null string shall  
5935 be considered to be longer than no match at all. For example, matching the BRE  
5936 “\(.\*\).\*” against “abcdef”, the subexpression “(\1)” is “abcdef”, and matching  
5937 the BRE “\ (a\*\) \*” against “bc”, the subexpression “(\1)” is the null string.

5938 When a multi-character collating element in a bracket expression (see Section 9.3.5 (on page  
5939 172)) is involved, the longest sequence shall be measured in characters consumed from the  
5940 string to be matched; that is, the collating element counts not as one element, but as the  
5941 number of characters it matches.

#### 5942 **BRE (ERE) matching a single character**

5943 A BRE or ERE that shall match either a single character or a single collating element.

5944 Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5 (on page  
5945 172)) can match a collating element.

#### 5946 **BRE (ERE) matching multiple characters**

5947 A BRE or ERE that shall match a concatenation of single characters or collating elements.

5948 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE)  
5949 special characters.

#### 5950 **invalid**

5951 This section uses the term “invalid” for certain constructs or conditions. Invalid REs shall  
5952 cause the utility or function using the RE to generate an error condition. When invalid is not  
5953 used, violations of the specified syntax or semantics for REs produce undefined results: this  
5954 may entail an error, enabling an extended syntax for that RE, or using the construct in error  
5955 as literal characters to be matched. For example, the BRE construct "`\{1,2,3\}`" does not  
5956 comply with the grammar. A conforming application cannot rely on it producing an error  
5957 nor matching the literal characters "`\{1,2,3\}`".

## 5958 **9.2 Regular Expression General Requirements**

5959 The requirements in this section shall apply to both basic and extended regular expressions.

5960 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)  
5961 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter  
5962 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that  
5963 is, zero or more characters followed by a <newline>. In the regular expression processing  
5964 described in IEEE Std 1003.1-2001, the <newline> is regarded as an ordinary character and both a  
5965 period and a non-matching list can match one. The Shell and Utilities volume of  
5966 IEEE Std 1003.1-2001 specifies within the individual descriptions of those standard utilities  
5967 employing regular expressions whether they permit matching of <newline>s; if not stated  
5968 otherwise, the use of literal <newline>s or any escape sequence equivalent produces undefined  
5969 results. Those utilities (like *grep*) that do not allow <newline>s to match are responsible for  
5970 eliminating any <newline> from strings before matching against the RE. The *regcomp()* function  
5971 in the System Interfaces volume of IEEE Std 1003.1-2001, however, can provide support for such  
5972 processing without violating the rules of this section.

5973 The interfaces specified in IEEE Std 1003.1-2001 do not permit the inclusion of a NUL character  
5974 in an RE or in the string to be matched. If during the operation of a standard utility a NUL is  
5975 included in the text designated to be matched, that NUL may designate the end of the text string  
5976 for the purposes of matching.

5977 When a standard utility or function that uses regular expressions specifies that pattern matching  
5978 shall be performed without regard to the case (uppercase or lowercase) of either data or  
5979 patterns, then when each character in the string is matched against the pattern, not only the  
5980 character, but also its case counterpart (if any), shall be matched. This definition of case-  
5981 insensitive processing is intended to allow matching of multi-character collating elements as  
5982 well as characters, as each character in the string is matched using both its cases. For example, in



5983 a locale where "Ch" is a multi-character collating element and where a matching list expression  
 5984 matches such elements, the RE "[[.Ch.]]" when matched against the string "char" is in  
 5985 reality matched against "ch", "Ch", "cH", and "CH".

5986 The implementation shall support any regular expression that does not exceed 256 bytes in  
 5987 length.

## 5988 **9.3 Basic Regular Expressions**

### 5989 **9.3.1 BREs Matching a Single Character or Collating Element**

5990 A BRE ordinary character, a special character preceded by a backslash, or a period shall match a  
 5991 single character. A bracket expression shall match a single character or a single collating  
 5992 element.

### 5993 **9.3.2 BRE Ordinary Characters**

5994 An ordinary character is a BRE that matches itself: any character in the supported character set,  
 5995 except for the BRE special characters listed in Section 9.3.3.

5996 The interpretation of an ordinary character preceded by a backslash ('\\') is undefined, except  
 5997 for:

- 5998 • The characters ' ) ', ' ( ', ' { ', and ' } '
- 5999 • The digits 1 to 9 inclusive (see Section 9.3.6 (on page 174))
- 6000 • A character inside a bracket expression

### 6001 **9.3.3 BRE Special Characters**

6002 A BRE special character has special properties in certain contexts. Outside those contexts, or  
 6003 when preceded by a backslash, such a character is a BRE that matches the special character itself.  
 6004 The BRE special characters and the contexts in which they have their special meaning are as  
 6005 follows:

6006 . [ \ The period, left-bracket, and backslash shall be special except when used in a bracket  
 6007 expression (see Section 9.3.5 (on page 172)). An expression containing a ' [ ' that is not  
 6008 preceded by a backslash and is not part of a bracket expression produces undefined  
 6009 results.

6010 \* The asterisk shall be special except when used:

- 6011 • In a bracket expression
- 6012 • As the first character of an entire BRE (after an initial ' ^ ', if any)
- 6013 • As the first character of a subexpression (after an initial ' ^ ', if any); see Section  
 6014 9.3.6 (on page 174)

6015 ^ The circumflex shall be special when used as:

- 6016 • An anchor (see Section 9.3.8 (on page 175))
- 6017 • The first character of a bracket expression (see Section 9.3.5 (on page 172))

6018 \$ The dollar sign shall be special when used as an anchor.

6019 **9.3.4 Periods in BREs**

6020 A period ( ' . ' ), when used outside a bracket expression, is a BRE that shall match any character  
 6021 in the supported character set except NUL.

6022 **9.3.5 RE Bracket Expression**

6023 A bracket expression (an expression enclosed in square brackets, "[ ]") is an RE that shall match  
 6024 a single collating element contained in the non-empty set of collating elements represented by  
 6025 the bracket expression.

6026 The following rules and definitions apply to bracket expressions:

6027 1. A bracket expression is either a matching list expression or a non-matching list expression.  
 6028 It consists of one or more expressions: collating elements, collating symbols, equivalence  
 6029 classes, character classes, or range expressions. The right-bracket ( ' ] ' ) shall lose its special  
 6030 meaning and represent itself in a bracket expression if it occurs first in the list (after an  
 6031 initial circumflex ( ' ^ ' ), if any). Otherwise, it shall terminate the bracket expression, unless  
 6032 it appears in a collating symbol (such as "[ . ] . ]") or is the ending right-bracket for a  
 6033 collating symbol, equivalence class, or character class. The special characters ' . ', ' \* ',  
 6034 ' [ ', and ' \ ' (period, asterisk, left-bracket, and backslash, respectively) shall lose their  
 6035 special meaning within a bracket expression.

6036 The character sequences "[ . ", "[ = ", and "[ : " (left-bracket followed by a period, equals-  
 6037 sign, or colon) shall be special inside a bracket expression and are used to delimit collating  
 6038 symbols, equivalence class expressions, and character class expressions. These symbols  
 6039 shall be followed by a valid expression and the matching terminating sequence ". ] ",  
 6040 "= ] ", or ": ] ", as described in the following items.

6041 2. A matching list expression specifies a list that shall match any single-character collating  
 6042 element in any of the expressions represented in the list. The first character in the list shall  
 6043 not be the circumflex; for example, "[ abc ]" is an RE that matches any of the characters  
 6044 ' a ', ' b ', or ' c '. It is unspecified whether a matching list expression matches a multi-  
 6045 character collating element that is matched by one of the expressions.

6046 3. A non-matching list expression begins with a circumflex ( ' ^ ' ), and specifies a list that  
 6047 shall match any single-character collating element except for the expressions represented  
 6048 in the list after the leading circumflex. For example, "[ ^ abc ]" is an RE that matches any  
 6049 character except the characters ' a ', ' b ', or ' c '. It is unspecified whether a non-matching  
 6050 list expression matches a multi-character collating element that is not matched by any of  
 6051 the expressions. The circumflex shall have this special meaning only when it occurs first in  
 6052 the list, immediately following the left-bracket.

6053 4. A collating symbol is a collating element enclosed within bracket-period ( "[ . " and ". ] ")  
 6054 delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 137).  
 6055 Conforming applications shall represent multi-character collating elements as collating  
 6056 symbols when it is necessary to distinguish them from a list of the individual characters  
 6057 that make up the multi-character collating element. For example, if the string "ch" is a  
 6058 collating element defined using the line:

6059 collating-element <ch-digraph> from "<c><h>"

6060 in the locale definition, the expression "[ [ . ch. ] ]" shall be treated as an RE containing  
 6061 the collating symbol ' ch ', while "[ ch ]" shall be treated as an RE matching ' c ' or ' h '.  
 6062 Collating symbols are recognized only inside bracket expressions. If the string is not a  
 6063 collating element in the current locale, the expression is invalid.

5. An equivalence class expression shall represent the set of collating elements belonging to an equivalence class, as described in Section 7.3.2.4 (on page 137). Only primary equivalence classes shall be recognized. The class shall be expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal ("[" and "=") delimiters. For example, if 'a', 'à', and 'â' belong to the same equivalence class, then "[[=a=]b]", "[[=à=]b]", and "[[=â=]b]" are each equivalent to "[aââb]". If the collating element does not belong to an equivalence class, the equivalence class expression shall be treated as a collating symbol.
6. A character class expression shall represent the union of two sets:
- The set of single-character collating elements whose characters belong to the character class, as defined in the *LC\_CTYPE* category in the current locale.
  - An unspecified set of multi-character collating elements.

All character classes specified in the current locale shall be recognized. A character class expression is expressed as a character class name enclosed within bracket-colon ("[" and ":"]") delimiters.

The following character class expressions shall be supported in all locales:

[ :alnum: ]	[ :cntrl: ]	[ :lower: ]	[ :space: ]
[ :alpha: ]	[ :digit: ]	[ :print: ]	[ :upper: ]
[ :blank: ]	[ :graph: ]	[ :punct: ]	[ :xdigit: ]

In addition, character class expressions of the form:

[ :name: ]

are recognized in those locales where the *name* keyword has been given a **charclass** definition in the *LC\_CTYPE* category.

7. In the POSIX locale, a range expression represents the set of collating elements that fall between two elements in the collation sequence, inclusive. In other locales, a range expression has unspecified behavior: strictly conforming applications shall not rely on whether the range expression is valid, or on the set of collating elements matched. A range expression shall be expressed as the starting point and the ending point separated by a hyphen ('-').

In the following, all examples assume the POSIX locale.

The starting range point and the ending range point shall be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. If the represented set of collating elements is empty, it is unspecified whether the expression matches nothing, or is treated as invalid.

The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

The hyphen character shall be treated as itself if it occurs first (after an initial '^', if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions "[ -ac]" and "[ac -]" are equivalent and match any of the characters 'a', 'c', or '-'; "[^ -ac]" and "[^ac -]" are equivalent and match any characters except 'a', 'c', or '-'; the expression "[ %-]" matches any of the characters between '%' and '-' inclusive; the expression "[ -@]" matches any of the characters between '-' and '@' inclusive; and the expression "[a -@]" is either invalid or equivalent to '@',

6109 because the letter 'a' follows the symbol '-' in the POSIX locale. To use a hyphen as the  
 6110 starting range point, it shall either come first in the bracket expression or be specified as a  
 6111 collating symbol; for example, "[[.-.]-0]", which matches either a right bracket or  
 6112 any character or collating element that collates between hyphen and 0, inclusive.

6113 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the  
 6114 '^', if any) and the '-' last within the bracket expression.

### 6115 9.3.6 BREs Matching Multiple Characters

6116 The following rules can be used to construct BREs matching multiple characters from BREs  
 6117 matching a single character:

- 6118 1. The concatenation of BREs shall match the concatenation of the strings matched by each  
 6119 component of the BRE.
- 6120 2. A subexpression can be defined within a BRE by enclosing it between the character pairs  
 6121 "\(" and "\)". Such a subexpression shall match whatever it would have matched  
 6122 without the "\(" and "\)", except that anchoring within subexpressions is optional  
 6123 behavior; see Section 9.3.8 (on page 175). Subexpressions can be arbitrarily nested.
- 6124 3. The back-reference expression '\n' shall match the same (possibly empty) string of  
 6125 characters as was matched by a subexpression enclosed between "\(" and "\)"  
 6126 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the  
 6127 *n*th subexpression (the one that begins with the *n*th "\(" from the beginning of the  
 6128 pattern and ends with the corresponding paired "\)"). The expression is invalid if less  
 6129 than *n* subexpressions precede the '\n'. For example, the expression "\(.\*)\1\$" matches a line  
 6130 consisting of two adjacent appearances of the same string, and the  
 6131 expression "\(a\)\*\1" fails to match 'a'. When the referenced subexpression matched  
 6132 more than one string, the back-referenced expression shall refer to the last matched string.  
 6133 If the subexpression referenced by the back-reference matches more than one string  
 6134 because of an asterisk ('\*') or an interval expression (see item (5)), the back-reference  
 6135 shall match the last (rightmost) of these strings.
- 6136 4. When a BRE matching a single character, a subexpression, or a back-reference is followed  
 6137 by the special character asterisk ('\*'), together with that asterisk it shall match what zero  
 6138 or more consecutive occurrences of the BRE would match. For example, "[ab]\*" and  
 6139 "[ab][ab]" are equivalent when matching the string "ab".
- 6140 5. When a BRE matching a single character, a subexpression, or a back-reference is followed  
 6141 by an interval expression of the format "{m}", "{m,}", or "{m,n}", together  
 6142 with that interval expression it shall match what repeated consecutive occurrences of the  
 6143 BRE would match. The values of *m* and *n* are decimal integers in the range 0  
 6144  $\leq m \leq n \leq \text{RE\_DUP\_MAX}$ , where *m* specifies the exact or minimum number of occurrences  
 6145 and *n* specifies the maximum number of occurrences. The expression "{m}" shall match  
 6146 exactly *m* occurrences of the preceding BRE, "{m,}" shall match at least *m* occurrences,  
 6147 and "{m,n}" shall match any number of occurrences between *m* and *n*, inclusive.

6148 For example, in the string "abababcccccccd" the BRE "c\{3\}" is matched by  
 6149 characters seven to nine, the BRE "\(ab\)\{4,\}" is not matched at all, and the BRE  
 6150 "c\{1,3\}d" is matched by characters ten to thirteen.

6151 The behavior of multiple adjacent duplication symbols ('\*' and intervals) produces undefined  
 6152 results.

6153 A subexpression repeated by an asterisk ('\*') or an interval expression shall not match a null  
 6154 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or

6155 minimum number of occurrences for the interval expression.

### 6156 9.3.7 BRE Precedence

6157 The order of precedence shall be as shown in the following table:

	BRE Precedence (from high to low)	
6158	Collation-related bracket symbols	[ = ] [ : : ] [ . . ]
6159	Escaped characters	\<special character>
6160	Bracket expression	[ ]
6161	Subexpressions/back-references	\ ( \ ) \ n
6162	Single-character-BRE duplication	* \{m,n\}
6163	Concatenation	
6164	Anchoring	^ \$

### 6166 9.3.8 BRE Expression Anchoring

6167 A BRE can be limited to matching strings that begin or end a line; this is called “anchoring”. The  
 6168 circumflex and dollar sign special characters shall be considered BRE anchors in the following  
 6169 contexts:

- 6170 1. A circumflex (‘^’) shall be an anchor when used as the first character of an entire BRE.  
 6171 The implementation may treat the circumflex as an anchor when used as the first character  
 6172 of a subexpression. The circumflex shall anchor the expression (or optionally  
 6173 subexpression) to the beginning of a string; only sequences starting at the first character of  
 6174 a string shall be matched by the BRE. For example, the BRE “^ab” matches “ab” in the  
 6175 string “abcdef”, but fails to match in the string “cdefab”. The BRE “\(^ab\)” may  
 6176 match the former string. A portable BRE shall escape a leading circumflex in a  
 6177 subexpression to match a literal circumflex.
- 6178 2. A dollar sign (‘\$’) shall be an anchor when used as the last character of an entire BRE.  
 6179 The implementation may treat a dollar sign as an anchor when used as the last character of  
 6180 a subexpression. The dollar sign shall anchor the expression (or optionally subexpression)  
 6181 to the end of the string being matched; the dollar sign can be said to match the end-of-  
 6182 string following the last character.
- 6183 3. A BRE anchored by both ‘^’ and ‘\$’ shall match only an entire string. For example, the  
 6184 BRE “^abcdef\$” matches strings consisting only of “abcdef”.

## 6185 9.4 Extended Regular Expressions

6186 The extended regular expression (ERE) notation and construction rules shall apply to utilities  
 6187 defined as using extended regular expressions; any exceptions to the following rules are noted in  
 6188 the descriptions of the specific utilities using EREs.

### 6189 9.4.1 EREs Matching a Single Character or Collating Element

6190 An ERE ordinary character, a special character preceded by a backslash, or a period shall match  
 6191 a single character. A bracket expression shall match a single character or a single collating  
 6192 element. An ERE matching a single character enclosed in parentheses shall match the same as  
 6193 the ERE without parentheses would have matched.

### 6194 9.4.2 ERE Ordinary Characters

6195 An ordinary character is an ERE that matches itself. An ordinary character is any character in the  
 6196 supported character set, except for the ERE special characters listed in Section 9.4.3. The  
 6197 interpretation of an ordinary character preceded by a backslash (' \ ') is undefined.

### 6198 9.4.3 ERE Special Characters

6199 An ERE special character has special properties in certain contexts. Outside those contexts, or  
 6200 when preceded by a backslash, such a character shall be an ERE that matches the special  
 6201 character itself. The extended regular expression special characters and the contexts in which  
 6202 they shall have their special meaning are as follows:

6203 . [ \ ( The period, left-bracket, backslash, and left-parenthesis shall be special except when  
 6204 used in a bracket expression (see Section 9.3.5 (on page 172)). Outside a bracket  
 6205 expression, a left-parenthesis immediately followed by a right-parenthesis produces  
 6206 undefined results.

6207 ) The right-parenthesis shall be special when matched with a preceding left-parenthesis,  
 6208 both outside a bracket expression.

6209 \* + ? { The asterisk, plus-sign, question-mark, and left-brace shall be special except when used  
 6210 in a bracket expression (see Section 9.3.5 (on page 172)). Any of the following uses  
 6211 produce undefined results:

- 6212 • If these characters appear first in an ERE, or immediately following a vertical-line,  
 6213 circumflex, or left-parenthesis
- 6214 • If a left-brace is not part of a valid interval expression (see Section 9.4.6 (on page  
 6215 177))

6216 | The vertical-line is special except when used in a bracket expression (see Section 9.3.5  
 6217 (on page 172)). A vertical-line appearing first or last in an ERE, or immediately  
 6218 following a vertical-line or a left-parenthesis, or immediately preceding a right-  
 6219 parenthesis, produces undefined results.

6220 ^ The circumflex shall be special when used as:

- 6221 • An anchor (see Section 9.4.9 (on page 178))
- 6222 • The first character of a bracket expression (see Section 9.3.5 (on page 172))

6223 \$ The dollar sign shall be special when used as an anchor.

6224 **9.4.4 Periods in EREs**

6225 A period ('.'), when used outside a bracket expression, is an ERE that shall match any  
 6226 character in the supported character set except NUL.

6227 **9.4.5 ERE Bracket Expression**

6228 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section  
 6229 9.3.5 (on page 172).

6230 **9.4.6 EREs Matching Multiple Characters**

6231 The following rules shall be used to construct EREs matching multiple characters from EREs  
 6232 matching a single character:

- 6233 1. A concatenation of EREs shall match the concatenation of the character sequences matched  
 6234 by each component of the ERE. A concatenation of EREs enclosed in parentheses shall  
 6235 match whatever the concatenation without the parentheses matches. For example, both the  
 6236 ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string  
 6237 "abcdefabcdef".
- 6238 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
 6239 the special character plus-sign ('+'), together with that plus-sign it shall match what one  
 6240 or more consecutive occurrences of the ERE would match. For example, the ERE  
 6241 "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde". And,  
 6242 "[ab]+" and "[ab][ab]\*" are equivalent.
- 6243 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
 6244 the special character asterisk ('\*'), together with that asterisk it shall match what zero or  
 6245 more consecutive occurrences of the ERE would match. For example, the ERE "b\*c"  
 6246 matches the first character in the string "cabbbbcde", and the ERE "b\*cd" matches the  
 6247 third to seventh characters in the string "cabbbbcdebbbbbcbdbc". And, "[ab]\*" and  
 6248 "[ab][ab]" are equivalent when matching the string "ab".
- 6249 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
 6250 the special character question-mark ('?'), together with that question-mark it shall match  
 6251 what zero or one consecutive occurrences of the ERE would match. For example, the ERE  
 6252 "b?c" matches the second character in the string "acabbbbcde".
- 6253 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
 6254 an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that  
 6255 interval expression it shall match what repeated consecutive occurrences of the ERE would  
 6256 match. The values of *m* and *n* are decimal integers in the range  $0 \leq m \leq n \leq \text{RE\_DUP\_MAX}$ ,  
 6257 where *m* specifies the exact or minimum number of occurrences and *n* specifies the  
 6258 maximum number of occurrences. The expression "{m}" matches exactly *m* occurrences  
 6259 of the preceding ERE, "{m,}" matches at least *m* occurrences, and "{m,n}" matches any  
 6260 number of occurrences between *m* and *n*, inclusive.

6261 For example, in the string "abababcccccd" the ERE "c{3}" is matched by characters  
 6262 seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

6263 The behavior of multiple adjacent duplication symbols ('+', '\*', '?', and intervals) produces  
 6264 undefined results.

6265 An ERE matching a single character repeated by an '\*', '?', or an interval expression shall not  
 6266 match a null expression unless this is the only match for the repetition or it is necessary to satisfy  
 6267 the exact or minimum number of occurrences for the interval expression.

### 9.4.7 ERE Alternation

Two EREs separated by the special character vertical-line ( ' | ' ) shall match a string that is matched by either. For example, the ERE "a ( (bc) | d )" matches the string "abc" and the string "ad". Single characters, or expressions matching single characters, separated by the vertical bar and enclosed in parentheses, shall be treated as an ERE matching a single character.

### 9.4.8 ERE Precedence

The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
Collation-related bracket symbols	[ = ] [ : : ] [ . . ]
Escaped characters	\<special character>
Bracket expression	[ ]
Grouping	( )
Single-character-ERE duplication	* + ? {m,n}
Concatenation	
Anchoring	^ \$
Alternation	

For example, the ERE "abba|cde" matches either the string "abba" or the string "cde" (rather than the string "abbade" or "abbcde", because concatenation has a higher order of precedence than alternation).

### 9.4.9 ERE Expression Anchoring

An ERE can be limited to matching strings that begin or end a line; this is called "anchoring". The circumflex and dollar sign special characters shall be considered ERE anchors when used anywhere outside a bracket expression. This shall have the following effects:

1. A circumflex ( ' ^ ' ) outside a bracket expression shall anchor the expression or subexpression it begins to the beginning of a string; such an expression or subexpression can match only a sequence starting at the first character of a string. For example, the EREs " ^ ab " and " ( ^ ab ) " match "ab" in the string "abcdef", but fail to match in the string "cdefab", and the ERE "a ^ b" is valid, but can never match because the 'a' prevents the expression " ^ b" from matching starting at the first character.
2. A dollar sign ( ' \$ ' ) outside a bracket expression shall anchor the expression or subexpression it ends to the end of a string; such an expression or subexpression can match only a sequence ending at the last character of a string. For example, the EREs " ef \$ " and " ( ef \$ ) " match "ef" in the string "abcdef", but fail to match in the string "cdefab", and the ERE "e \$ f" is valid, but can never match because the 'f' prevents the expression "e \$ " from matching ending at the last character.



## 6303 9.5 Regular Expression Grammar

6304 Grammars describing the syntax of both basic and extended regular expressions are presented in  
6305 this section. The grammar takes precedence over the text. See the Shell and Utilities volume of  
6306 IEEE Std 1003.1-2001, Section 1.10, Grammar Conventions.

### 6307 9.5.1 BRE/ERE Grammar Lexical Conventions

6308 The lexical conventions for regular expressions are as described in this section.

6309 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6310 The following tokens are processed (in addition to those string constants shown in the  
6311 grammar):

#### 6312 COLL\_ELEM\_SINGLE

6313 Any single-character collating element, unless it is a **META\_CHAR**.

#### 6314 COLL\_ELEM\_MULTI

6315 Any multi-character collating element.

#### 6316 BACKREF

6317 Applicable only to basic regular expressions. The character string  
consisting of `'\'` followed by a single-digit numeral, `'1'` to `'9'`.

#### 6318 DUP\_COUNT

6319 Represents a numeric constant. It shall be an integer in the range  $0 \leq \text{DUP\_COUNT} \leq \{\text{RE\_DUP\_MAX}\}$ . This token is only recognized when  
6320 the context of the grammar requires it. At all other times, digits not  
6321 preceded by `'\'` are treated as **ORD\_CHAR**.

#### 6322 META\_CHAR

6323 One of the characters:

6323 `^` When found first in a bracket expression

6324 `-` When found anywhere but first (after an initial `'^'`, if any) or  
6325 last in a bracket expression, or as the ending range point in a  
6326 range expression

6327 `]` When found anywhere but first (after an initial `'^'`, if any) in a  
6328 bracket expression

#### 6329 L\_ANCHOR

6330 Applicable only to basic regular expressions. The character `'^'` when it  
6331 appears as the first character of a basic regular expression and when not  
6332 **QUOTED\_CHAR**. The `'^'` may be recognized as an anchor elsewhere;  
see Section 9.3.8 (on page 175).

#### 6333 ORD\_CHAR

A character, other than one of the special characters in **SPEC\_CHAR**.

#### 6334 QUOTED\_CHAR

In a BRE, one of the character sequences:

6335 `\^` `\.` `\*` `\[` `\$` `\\`

6336 In an ERE, one of the character sequences:

6337 `\^` `\.` `\[` `\$` `\(` `\)` `\\`

6338 `\*` `\+` `\?` `\{` `\\`

#### 6339 R\_ANCHOR

6340 (Applicable only to basic regular expressions.) The character `'$'` when it  
6341 appears as the last character of a basic regular expression and when not  
6342 **QUOTED\_CHAR**. The `'$'` may be recognized as an anchor elsewhere;  
see Section 9.3.8 (on page 175).

6343	<b>SPEC_CHAR</b>	For basic regular expressions, one of the following special characters:
6344	.	Anywhere outside bracket expressions
6345	\	Anywhere outside bracket expressions
6346	[	Anywhere outside bracket expressions
6347	^	When used as an anchor (see Section 9.3.8 (on page 175)) or
6348		when first in a bracket expression
6349	\$	When used as an anchor
6350	*	Anywhere except first in an entire RE, anywhere in a bracket
6351		expression, directly following "\(", directly following an
6352		anchoring '^'
6353		For extended regular expressions, shall be one of the following special
6354		characters found anywhere outside bracket expressions:
6355	^ . [ \$ ( )	
6356	* + ? { \	
6357		(The close-parenthesis shall be considered special in this context only if
6358		matched with a preceding open-parenthesis.)

## 6359 9.5.2 RE and Bracket Expression Grammar

6360 This section presents the grammar for basic regular expressions, including the bracket  
6361 expression grammar that is common to both BREs and EREs.

```

6362 %token    ORD_CHAR QUOTED_CHAR DUP_COUNT
6363 %token    BACKREF L_ANCHOR R_ANCHOR
6364 %token    Back_open_paren Back_close_paren
6365 /*      '\('      '\)'      */
6366 %token    Back_open_brace Back_close_brace
6367 /*      '\{'      '\}'      */
6368 /* The following tokens are for the Bracket Expression
6369    grammar common to both RES and ERES. */
6370 %token    COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6371 %token    Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6372 /*      '['      '='      '['      '.'      ':'      ':'      */
6373 %token    class_name
6374 /* class_name is a keyword to the LC_CTYPE locale category */
6375 /* (representing a character class) in the current locale */
6376 /* and is only recognized between [: and :] */
6377 %start    basic_reg_exp
6378 %%
6379 /* -----
6380    Basic Regular Expression
6381    -----
6382    */
6383 basic_reg_exp :      RE_expression

```

```

6384         | L_ANCHOR
6385         |                                     R_ANCHOR
6386         | L_ANCHOR R_ANCHOR
6387         | L_ANCHOR RE_expression
6388         | RE_expression R_ANCHOR
6389         | L_ANCHOR RE_expression R_ANCHOR
6390         ;
6391 RE_expression : simple_RE
6392         | RE_expression simple_RE
6393         ;
6394 simple_RE : nondupl_RE
6395         | nondupl_RE RE_dupl_symbol
6396         ;
6397 nondupl_RE : one_char_or_coll_elem_RE
6398         | Back_open_paren RE_expression Back_close_paren
6399         | BACKREF
6400         ;
6401 one_char_or_coll_elem_RE : ORD_CHAR
6402         | QUOTED_CHAR
6403         | '.'
6404         | bracket_expression
6405         ;
6406 RE_dupl_symbol : '*'
6407         | Back_open_brace DUP_COUNT Back_close_brace
6408         | Back_open_brace DUP_COUNT ',' Back_close_brace
6409         | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6410         ;
6411 /* -----
6412    Bracket Expression
6413    -----
6414 */
6415 bracket_expression : '[' matching_list ']'
6416         | '[' nonmatching_list ']'
6417         ;
6418 matching_list : bracket_list
6419         ;
6420 nonmatching_list : '^' bracket_list
6421         ;
6422 bracket_list : follow_list
6423         | follow_list '-'
6424         ;
6425 follow_list : expression_term
6426         | follow_list expression_term
6427         ;
6428 expression_term : single_expression
6429         | range_expression
6430         ;
6431 single_expression : end_range
6432         | character_class
6433         | equivalence_class
6434         ;
6435 range_expression : start_range end_range

```

```

6436         | start_range '-'
6437         ;
6438     start_range      : end_range '-'
6439         ;
6440     end_range        : COLL_ELEM_SINGLE
6441         | collating_symbol
6442         ;
6443     collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
6444         | Open_dot COLL_ELEM_MULTI Dot_close
6445         | Open_dot META_CHAR Dot_close
6446         ;
6447     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6448         | Open_equal COLL_ELEM_MULTI Equal_close
6449         ;
6450     character_class : Open_colon class_name Colon_close
6451         ;

```

6452 The BRE grammar does not permit **L\_ANCHOR** or **R\_ANCHOR** inside `"\"` and `"\"` (which  
6453 implies that `'^'` and `'$'` are ordinary characters). This reflects the semantic limits on the  
6454 application, as noted in Section 9.3.8 (on page 175). Implementations are permitted to extend the  
6455 language to interpret `'^'` and `'$'` as anchors in these locations, and as such, conforming  
6456 applications cannot use unescaped `'^'` and `'$'` in positions inside `"\"` and `"\"` that might  
6457 be interpreted as anchors.

### 6458 9.5.3 ERE Grammar

6459 This section presents the grammar for extended regular expressions, excluding the bracket  
6460 expression grammar.

6461 **Note:** The bracket expression grammar and the associated **%token** lines are identical between BREs  
6462 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6463 %token  ORD_CHAR QUOTED_CHAR DUP_COUNT
6464 %start  extended_reg_exp
6465 %%
6466 /* -----
6467    Extended Regular Expression
6468    -----
6469 */
6470 extended_reg_exp      :                ERE_branch
6471         | extended_reg_exp '|' ERE_branch
6472         ;
6473 ERE_branch            :                ERE_expression
6474         | ERE_branch ERE_expression
6475         ;
6476 ERE_expression        : one_char_or_coll_elem_ERE
6477         | '^'
6478         | '$'
6479         | '(' extended_reg_exp ')'
6480         | ERE_expression ERE_dupl_symbol
6481         ;
6482 one_char_or_coll_elem_ERE : ORD_CHAR
6483         | QUOTED_CHAR
6484         | '.'

```

```

6485             | bracket_expression
6486             ;
6487 ERE_dupl_symbol : '*'
6488             | '+'
6489             | '?'
6490             | '{' DUP_COUNT '}'
6491             | '{' DUP_COUNT ',' '}'
6492             | '{' DUP_COUNT ',' DUP_COUNT '}'
6493             ;

```

The ERE grammar does not permit several constructs that previous sections specify as having undefined results:

- **ORD\_CHAR** preceded by '`\`'
- One or more *ERE\_dupl\_symbols* appearing first in an ERE, or immediately following '`|`', '`^`', or '`(`'
- '`{`' not part of a valid *ERE\_dupl\_symbol*
- '`|`' appearing first or last in an ERE, or immediately following '`|`' or '`(`', or immediately preceding '`)`'

Implementations are permitted to extend the language to allow these. Conforming applications cannot use such constructs.



# Directory Structure and Devices

6505

## 10.1 Directory Structure and Files

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

**/** The root directory.

**/dev** Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described:

**/tmp** A directory made available for applications that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable:

**/dev/null** An infinite data source and data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

**/dev/tty** In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for applications that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

**/dev/console** The **/dev/console** file is a generic name given to the system console (see Section 3.382 (on page 88)). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.

## 10.2 Output Devices and Terminal Types

The utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. IEEE Std 1003.1-2001 states which features are optional for certain classes of terminals in the individual utility description sections. The implementation shall document in the system documentation which terminal types it supports and which of these features and utilities are not supported by each terminal. 2 2

When a feature or utility is not supported on a specific terminal type, as allowed by IEEE Std 1003.1-2001, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that inform the user that the terminal type lacks the appropriate capability.

IEEE Std 1003.1-2001 uses a notational convention based on historical practice that identifies some of the control characters defined in Section 7.3.1 (on page 126) in a manner easily remembered by users on many terminals. The correspondence between this “<control>-char” notation and the actual control characters is shown in the following table. When IEEE Std 1003.1-2001 refers to a character by its <control>-name, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the non-alphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal are interpreted by the system as representing the special characters.

Table 10-1 Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y	<EM>	<EM>
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[	<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-]	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-^	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-_	<US>	<US>
<control>-P	<DLE>	<DLE>	<control>-?	<DEL>	<DEL>

**Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.



# General Terminal Interface

6573

6574 This chapter describes a general terminal interface that shall be provided. It shall be supported  
 6575 on any asynchronous communications ports if the implementation provides them. It is  
 6576 implementation-defined whether it supports network connections or synchronous ports, or  
 6577 both.

## 6578 11.1 Interface Characteristics

### 6579 11.1.1 Opening a Terminal Device File

6580 When a terminal device file is opened, it normally causes the thread to wait until a connection is  
 6581 established. In practice, application programs seldom open these files; they are opened by  
 6582 special programs and become an application's standard input, output, and error files.

6583 As described in *open()*, opening a terminal device file with the *O\_NONBLOCK* flag clear shall  
 6584 cause the thread to block until the terminal device is ready and available. If *CLOCAL* mode is  
 6585 not set, this means blocking until a connection is established. If *CLOCAL* mode is set in the  
 6586 terminal, or the *O\_NONBLOCK* flag is specified in the *open()*, the *open()* function shall return a  
 6587 file descriptor without waiting for a connection to be established.

### 6588 11.1.2 Process Groups

6589 A terminal may have a foreground process group associated with it. This foreground process  
 6590 group plays a special role in handling signal-generating input characters, as discussed in Section  
 6591 11.1.9 (on page 191).

6592 A command interpreter process supporting job control can allocate the terminal to different jobs,  
 6593 or process groups, by placing related processes in a single process group and associating this  
 6594 process group with the terminal. A terminal's foreground process group may be set or examined  
 6595 by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The  
 6596 terminal interface aids in this allocation by restricting access to the terminal by processes that are  
 6597 not in the current process group; see Section 11.1.4 (on page 188).

6598 When there is no longer any process whose process ID or process group ID matches the  
 6599 foreground process group ID, the terminal shall have no foreground process group. It is  
 6600 unspecified whether the terminal has a foreground process group when there is a process whose  
 6601 process ID matches the foreground process group ID, but whose process group ID does not. No  
 6602 actions defined in IEEE Std 1003.1-2001, other than allocation of a controlling terminal or a  
 6603 successful call to *tcsetpgrp()*, shall cause a process group to become the foreground process  
 6604 group of the terminal.

### 6605 11.1.3 The Controlling Terminal

6606 A terminal may belong to a process as its controlling terminal. Each process of a session that has  
 6607 a controlling terminal has the same controlling terminal. A terminal may be the controlling  
 6608 terminal for at most one session. The controlling terminal for a session is allocated by the session  
 6609 leader in an implementation-defined manner. If a session leader has no controlling terminal, and  
 6610 opens a terminal device file that is not already associated with a session without using the  
 6611 O\_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the  
 6612 controlling terminal of the session leader. If a process which is not a session leader opens a  
 6613 terminal file, or the O\_NOCTTY option is used on *open()*, then that terminal shall not become  
 6614 the controlling terminal of the calling process. When a controlling terminal becomes associated  
 6615 with a session, its foreground process group shall be set to the process group of the session  
 6616 leader.

6617 The controlling terminal is inherited by a child process during a *fork()* function call. A process  
 6618 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;  
 6619 other processes remaining in the old session that had this terminal as their controlling terminal  
 6620 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in  
 6621 the current session) associated with the controlling terminal, it is unspecified whether all  
 6622 processes that had that terminal as their controlling terminal cease to have any controlling  
 6623 terminal. Whether and how a session leader can reacquire a controlling terminal after the  
 6624 controlling terminal has been relinquished in this fashion is unspecified. A process does not  
 6625 relinquish its controlling terminal simply by closing all of its file descriptors associated with the  
 6626 controlling terminal if other processes continue to have it open.

6627 When a controlling process terminates, the controlling terminal is dissociated from the current  
 6628 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by  
 6629 other processes in the earlier session may be denied, with attempts to access the terminal treated  
 6630 as if a modem disconnect had been sensed.

### 6631 11.1.4 Terminal Access Control

6632 If a process is in the foreground process group of its controlling terminal, read operations shall  
 6633 be allowed, as described in Section 11.1.5 (on page 189). Any attempts by a process in a  
 6634 background process group to read from its controlling terminal cause its process group to be  
 6635 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is  
 6636 ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is  
 6637 orphaned, the *read()* shall return -1, with *errno* set to [EIO] and no signal shall be sent. The  
 6638 default action of the SIGTTIN signal shall be to stop the process to which it is sent. See  
 6639 <signal.h>.

6640 If a process is in the foreground process group of its controlling terminal, write operations shall  
 6641 be allowed as described in Section 11.1.8 (on page 191). Attempts by a process in a background  
 6642 process group to write to its controlling terminal shall cause the process group to be sent a  
 6643 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if  
 6644 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is  
 6645 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the  
 6646 process group of the writing process is orphaned, and the writing process is not ignoring or  
 6647 blocking the SIGTTOU signal, the *write()* shall return -1, with *errno* set to [EIO] and no signal  
 6648 shall be sent.

6649 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that  
 6650 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set  
 6651 (see Section 11.2.5 (on page 197), *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcsetattr()*, and  
 6652 *tcsetpgrp()*).

### 6653 11.1.5 Input Processing and Reading Data

6654 A terminal device associated with a terminal device file may operate in full-duplex mode, so that  
 6655 data may arrive even while output is occurring. Each terminal device file has an input queue  
 6656 associated with it, into which incoming data is stored by the system before being read by a  
 6657 process. The system may impose a limit, {MAX\_INPUT}, on the number of bytes that may be  
 6658 stored in the input queue. The behavior of the system when this limit is exceeded is  
 6659 implementation-defined.

6660 Two general kinds of input processing are available, determined by whether the terminal device  
 6661 file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 and  
 6662 Section 11.1.7 (on page 190). Additionally, input characters are processed according to the *c\_iflag*  
 6663 (see Section 11.2.2 (on page 193)) and *c\_lflag* (see Section 11.2.5 (on page 197)) fields. Such  
 6664 processing can include “echoing”, which in general means transmitting input characters  
 6665 immediately back to the terminal when they are received from the terminal. This is useful for  
 6666 terminals that can operate in full-duplex mode.

6667 The manner in which data is provided to a process reading from a terminal device file is  
 6668 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether  
 6669 or not the O\_NONBLOCK flag is set by *open()* or *fcntl()*.

6670 If the O\_NONBLOCK flag is clear, then the read request shall be blocked until data is available  
 6671 or a signal has been received. If the O\_NONBLOCK flag is set, then the read request shall be  
 6672 completed, without blocking, in one of three ways:

- 6673 1. If there is enough data available to satisfy the entire request, the *read()* shall complete  
 6674 successfully and shall return the number of bytes read.
- 6675 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete  
 6676 successfully, having read as much data as possible, and shall return the number of bytes it  
 6677 was able to read.
- 6678 3. If there is no data available, the *read()* shall return -1, with *errno* set to [EAGAIN].

6679 When data is available depends on whether the input processing mode is canonical or non-  
 6680 canonical. Section 11.1.6 and Section 11.1.7 (on page 190) describe each of these input processing  
 6681 modes.

### 6682 11.1.6 Canonical Mode Input Processing

6683 In canonical mode input processing, terminal input is processed in units of lines. A line is  
 6684 delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL)  
 6685 character. See Section 11.1.9 (on page 191) for more information on EOF and EOL. This means  
 6686 that a read request shall not return until an entire line has been typed or a signal has been  
 6687 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall  
 6688 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even  
 6689 one, may be requested in a *read()* without losing information.

6690 If {MAX\_CANON} is defined for this terminal device, it shall be a limit on the number of bytes  
 6691 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If  
 6692 {MAX\_CANON} is not defined, there shall be no such limit; see *pathconf()*.

6693 Erase and kill processing occur when either of two special characters, the ERASE and KILL  
 6694 characters (see Section 11.1.9 (on page 191)), is received. This processing shall affect data in the  
 6695 input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited  
 6696 data makes up the current line. The ERASE character shall delete the last character in the current  
 6697 line, if there is one. The KILL character shall delete all data in the current line, if there is any. The  
 6698 ERASE and KILL characters shall have no effect if there is no data in the current line. The ERASE

6699 and KILL characters themselves shall not be placed in the input queue.

### 6700 11.1.7 Non-Canonical Mode Input Processing

6701 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and  
6702 kill processing shall not occur. The values of the MIN and TIME members of the `c_cc` array are  
6703 used to determine how to process the bytes received. IEEE Std 1003.1-2001 does not specify  
6704 whether the setting of O\_NONBLOCK takes precedence over MIN or TIME settings. Therefore,  
6705 if O\_NONBLOCK is set, `read()` may return immediately, regardless of the setting of MIN or  
6706 TIME. Also, if no data is available, `read()` may either return 0, or return -1 with `errno` set to  
6707 [EAGAIN].

6708 MIN represents the minimum number of bytes that should be received when the `read()` function  
6709 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and  
6710 short-term data transmissions. If MIN is greater than {MAX\_INPUT}, the response to the request  
6711 is undefined. The four possible values for MIN and TIME and their interactions are described  
6712 below.

#### 6713 Case A: MIN>0, TIME>0

6714 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is  
6715 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction  
6716 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall  
6717 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer  
6718 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN  
6719 bytes are received, the characters received to that point shall be returned to the user. Note that if  
6720 TIME expires at least one byte shall be returned because the timer would not have been enabled  
6721 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and  
6722 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in  
6723 the buffer at the time of the `read()`, the result shall be as if data has been received immediately  
6724 after the `read()`.

#### 6725 Case B: MIN>0, TIME=0

6726 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A  
6727 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall  
6728 block until MIN bytes are received), or a signal is received. A program that uses case B to read  
6729 record-based terminal I/O may block indefinitely in the read operation.

#### 6730 Case C: MIN=0, TIME>0

6731 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read  
6732 timer that shall be activated as soon as the `read()` function is processed. A read shall be satisfied  
6733 as soon as a single byte is received or the read timer expires. Note that in case C if the timer  
6734 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be  
6735 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely  
6736 waiting for a byte; if no byte is received within TIME\*0.1 seconds after the read is initiated, the  
6737 `read()` shall return a value of zero, having read no data. If data is in the buffer at the time of the  
6738 `read()`, the timer shall be started as if data has been received immediately after the `read()`.

6739           **Case D: MIN=0, TIME=0**

6740           The minimum of either the number of bytes requested or the number of bytes currently available  
6741           shall be returned without waiting for more bytes to be input. If no characters are available, *read()*  
6742           shall return a value of zero, having read no data.

6743   **11.1.8 Writing Data and Output Processing**

6744           When a process writes one or more bytes to a terminal device file, they are processed according  
6745           to the *c\_oflag* field (see Section 11.2.3 (on page 194)). The implementation may provide a  
6746           buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have  
6747           been scheduled for transmission to the device, but the transmission has not necessarily  
6748           completed. See *write()* for the effects of *O\_NONBLOCK* on *write()*.

6749   **11.1.9 Special Characters**

6750           Certain characters have special functions on input or output or both. These functions are  
6751           summarized as follows:

6752           INTR    Special character on input, which is recognized if the ISIG flag is set. Generates a  
6753                      SIGINT signal which is sent to all processes in the foreground process group for which  
6754                      the terminal is the controlling terminal. If ISIG is set, the INTR character shall be  
6755                      discarded when processed.

6756           QUIT   Special character on input, which is recognized if the ISIG flag is set. Generates a  
6757                      SIGQUIT signal which is sent to all processes in the foreground process group for  
6758                      which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be  
6759                      discarded when processed.

6760           ERASE   Special character on input, which is recognized if the ICANON flag is set. Erases the  
6761                      last character in the current line; see Section 11.1.6 (on page 189). It shall not erase  
6762                      beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is  
6763                      set, the ERASE character shall be discarded when processed.

6764           KILL    Special character on input, which is recognized if the ICANON flag is set. Deletes the  
6765                      entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL  
6766                      character shall be discarded when processed.

6767           EOF      Special character on input, which is recognized if the ICANON flag is set. When  
6768                      received, all the bytes waiting to be read are immediately passed to the process without  
6769                      waiting for a newline, and the EOF is discarded. Thus, if there are no bytes waiting  
6770                      (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be  
6771                      returned from the *read()*, representing an end-of-file indication. If ICANON is set, the  
6772                      EOF character shall be discarded when processed.

6773           NL      Special character on input, which is recognized if the ICANON flag is set. It is the line  
6774                      delimiter newline. It cannot be changed.

6775           EOL     Special character on input, which is recognized if the ICANON flag is set. It is an  
6776                      additional line delimiter, like NL.

6777           SUSP    If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be  
6778                      sent to all processes in the foreground process group for which the terminal is the  
6779                      controlling terminal, and the SUSP character shall be discarded when processed.

6780           STOP    Special character on both input and output, which is recognized if the IXON (output  
6781                      control) or IXOFF (input control) flag is set. Can be used to suspend output  
6782                      temporarily. It is useful with CRT terminals to prevent output from disappearing

6783 before it can be read. If IXON is set, the STOP character shall be discarded when  
 6784 processed.

6785 **START** Special character on both input and output, which is recognized if the IXON (output  
 6786 control) or IXOFF (input control) flag is set. Can be used to resume output that has  
 6787 been suspended by a STOP character. If IXON is set, the START character shall be  
 6788 discarded when processed.

6789 **CR** Special character on input, which is recognized if the ICANON flag is set; it is the  
 6790 carriage-return character. When ICANON and ICRNL are set and IGNCR is not set,  
 6791 this character shall be translated into an NL, and shall have the same effect as an NL  
 6792 character.

6793 The NL and CR characters cannot be changed. It is implementation-defined whether the START  
 6794 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and  
 6795 SUSP shall be changeable to suit individual tastes. Special character functions associated with  
 6796 changeable special control characters can be disabled individually.

6797 If two or more special characters have the same value, the function performed when that  
 6798 character is received is undefined.

6799 A special character is recognized not only by its value, but also by its context; for example, an  
 6800 implementation may support multi-byte sequences that have a meaning different from the  
 6801 meaning of the bytes when considered individually. Implementations may also support  
 6802 additional single-byte functions. These implementation-defined multi-byte or single-byte  
 6803 functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without  
 6804 interpretation, except as required to recognize the special characters defined in this section.

6805 **XSI** If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding '`\`'  
 6806 character, in which case no special function shall occur.

#### 6807 **11.1.10 Modem Disconnect**

6808 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if  
 6809 CLOCAL is not set in the *c\_cflag* field for the terminal (see Section 11.2.4 (on page 196)), the  
 6810 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling  
 6811 terminal. Unless other arrangements have been made, this shall cause the controlling process to  
 6812 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of  
 6813 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-  
 6814 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*  
 6815 also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent  
 6816 *write()* to the terminal device shall return -1, with *errno* set to [EIO], until the device is closed.

#### 6817 **11.1.11 Closing a Terminal Device File**

6818 The last process to close a terminal device file shall cause any output to be sent to the device and  
 6819 any input to be discarded. If HUPCL is set in the control modes and the communications port  
 6820 supports a disconnect function, the terminal device shall perform a disconnect.

## 11.2 Parameters that Can be Set

### 11.2.1 The `termios` Structure

Routines that need to control certain terminal I/O characteristics shall do so by using the **termios** structure as defined in the `<termios.h>` header. The members of this structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
<code>tcflag_t</code>	NCCS	<code>c_iflag</code>	Input modes.
<code>tcflag_t</code>		<code>c_oflag</code>	Output modes.
<code>tcflag_t</code>		<code>c_cflag</code>	Control modes.
<code>tcflag_t</code>		<code>c_lflag</code>	Local modes.
<code>cc_t</code>		<code>c_cc[]</code>	Control characters.

The types `tcflag_t` and `cc_t` are defined in the `<termios.h>` header. They shall be unsigned integer types.

### 11.2.2 Input Modes

Values of the `c_iflag` field describe the basic terminal input control, and are composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in `<termios.h>`:

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

In the context of asynchronous serial data transmission, a break condition shall be defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission, the definition of a break condition is implementation-defined.

If `IGNBRK` is set, a break condition detected on input shall be ignored; that is, not put on the input queue and therefore not read by any process. If `IGNBRK` is not set and `BRKINT` is set, the break condition shall flush the input and output queues, and if the terminal is the controlling terminal of a foreground process group, the break condition shall generate a single `SIGINT` signal to that foreground process group. If neither `IGNBRK` nor `BRKINT` is set, a break condition shall be read as a single 0x00, or if `PARMRK` is set, as 0xff 0x00 0x00.

If `IGNPAR` is set, a byte with a framing or parity error (other than break) shall be ignored.

6865 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than  
 6866 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is  
 6867 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid  
 6868 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff  
 6869 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be  
 6870 given to the application as a single byte 0x00.

6871 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking  
 6872 shall be disabled, allowing output parity generation without input parity errors. Note that  
 6873 whether input parity checking is enabled or disabled is independent of whether parity detection  
 6874 is enabled or disabled (see Section 11.2.4 (on page 196)). If parity detection is enabled but input  
 6875 parity checking is disabled, the hardware to which the terminal is connected shall recognize the  
 6876 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

6877 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits  
 6878 shall be processed.

6879 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a  
 6880 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a  
 6881 received CR character shall be translated into an NL character.

6882 XSI If IXANY is set, any input character shall restart output that has been suspended.

6883 If IXON is set, start/stop output control shall be enabled. A received STOP character shall  
 6884 suspend output and a received START character shall restart output. When IXON is set, START  
 6885 and STOP characters are not read, but merely perform flow control functions. When IXON is not  
 6886 set, the START and STOP characters shall be read.

6887 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP  
 6888 characters, which are intended to cause the terminal device to stop transmitting data, as needed  
 6889 to prevent the input queue from overflowing and causing implementation-defined behavior, and  
 6890 shall transmit START characters, which are intended to cause the terminal device to resume  
 6891 transmitting data, as soon as the device can continue transmitting data without risk of  
 6892 overflowing the input queue. The precise conditions under which STOP and START characters  
 6893 are transmitted are implementation-defined.

6894 The initial input control value after *open()* is implementation-defined.

### 6895 11.2.3 Output Modes

6896 The *c\_oflag* field specifies the terminal interface's treatment of output, and is composed of the  
 6897 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
 6898 symbols in the following table are defined in `<termios.h>`:



6899  
6900  
6901  
6902 XSI  
6903  
6904  
6905  
6906  
6907  
6908  
6909  
6910  
6911  
6912  
6913  
6914  
6915  
6916  
6917  
6918  
6919  
6920  
6921  
6922  
6923  
6924  
6925  
6926  
6927  
6928  
6929

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

6930  
6931  
6932

If OPOST is set, output data shall be post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters shall be transmitted without change.

6933 XSI  
6934  
6935  
6936  
6937  
6938  
6939

If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

6940  
6941  
6942  
6943  
6944

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character shall be DEL; otherwise, NUL.

6945

If a form-feed or vertical-tab delay is specified, it shall last for about 2 seconds.

6946  
6947

Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall transmit two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be about 0.10 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two fill characters shall be transmitted for any delay.

Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be transmitted.

The actual delays depend on line speed and system load.

The initial output control value after *open()* is implementation-defined.

## 11.2.4 Control Modes

The *c\_cflag* field describes the hardware control of the terminal, and is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in `<termios.h>`; not all values specified are required to be supported by the underlying hardware:

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

In addition, the input and output baud rates are stored in the **termios** structure. The symbols in the following table are defined in `<termios.h>`. Not all values specified are required to be supported by the underlying hardware.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

The following functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*. The effects on the terminal device shall not become effective and not all errors need be detected until the *tcsetattr()* function is successfully called.

The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-

6993 order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read.  
 6994 CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used;  
 6995 otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

6996 If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

6997 If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to  
 6998 each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity  
 6999 shall be used.

7000 If HUPCL is set, the modem control lines for the port shall be lowered when the last process  
 7001 with the port open closes the port or the process terminates. The modem connection shall be  
 7002 broken.

7003 If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If  
 7004 CLOCAL is clear, the modem status lines shall be monitored.

7005 Under normal circumstances, a call to the *open()* function shall wait for the modem connection  
 7006 to complete. However, if the O\_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set,  
 7007 the *open()* function shall return immediately without waiting for the connection.

7008 If the object for which the control modes are set is not an asynchronous serial connection, some  
 7009 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a  
 7010 network connection to a terminal on another host, the baud rate need not be set on the  
 7011 connection between that terminal and the machine to which it is directly connected.

7012 The initial hardware control value after *open()* is implementation-defined.

### 7013 11.2.5 Local Modes

7014 The *c\_lflag* field of the argument structure is used to control various functions. It is composed of  
 7015 the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
 7016 symbols in this table are defined in *<termios.h>*; not all values specified are required to be  
 7017 supported by the underlying hardware:

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

7029 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input  
 7030 characters shall not be echoed.

7031 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if  
 7032 possible, the last character in the current line from the display. If there is no character to erase, an  
 7033 implementation may echo an indication that this was the case, or do nothing.

7034 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the  
 7035 line from the display or shall echo the newline character after the KILL character.

7036 If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.

7037 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit  
7038 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as  
7039 described in Section 11.1.6 (on page 189).

7040 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall  
7041 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired  
7042 between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 190) for  
7043 more details.

7044 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is  
7045 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.  
7046 If IEXTEN is not set, implementation-defined functions shall not be recognized and the  
7047 corresponding input characters are processed as described for ICANON, ISIG, IXON, and  
7048 IXOFF.

7049 If ISIG is set, each input character shall be checked against the special control characters INTR,  
7050 QUIT, and SUSP. If an input character matches one of these control characters, the function  
7051 associated with that character shall be performed. If ISIG is not set, no checking shall be done.  
7052 Thus these special input functions are possible only if ISIG is set.

7053 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,  
7054 QUIT, and SUSP characters shall not be done.

7055 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to  
7056 write to its controlling terminal if it is not in the foreground process group for that terminal. This  
7057 signal, by default, stops the members of the process group. Otherwise, the output generated by  
7058 that process shall be output to the current output stream. Processes that are blocking or ignoring  
7059 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall  
7060 not be sent.

7061 The initial local control value after *open()* is implementation-defined.

#### 7062 11.2.6 Special Control Characters

7063 The special control character values shall be defined by the array *c\_cc*. The subscript name and  
7064 description for each element in both canonical and non-canonical modes are as follows:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF	VINTR	EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR		INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

Implementations that do not support changing the START and STOP characters may ignore the character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when `tcsetattr()` is called, but shall return the value in use when `tcgetattr()` is called.

The initial values of all control characters are implementation-defined.

If the value of one of the changeable special control characters (see Section 11.1.9 (on page 191)) is `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special meaning for the VMIN and VTIME entries of the `c_cc` array.

7090

# Utility Conventions

7091

## 12.1 Utility Argument Syntax

This section describes the argument syntax of the standard utilities and introduces terminology used throughout IEEE Std 1003.1-2001 for describing the arguments processed by the utilities.

Within IEEE Std 1003.1-2001, a special notation is used for describing the syntax of a utility's arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated by this example (see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1, Simple Commands):

```
utility_name[-a][-b][-c option_argument]
          [-d|-e][-foption_argument][operand...]
```

The notation used for the SYNOPSIS sections imposes requirements on the implementors of the standard utilities and provides a simple reference for the application developer or system user.

1. The utility in the example is named *utility\_name*. It is followed by options, option-arguments, and operands. The arguments that consist of hyphens and single letters or digits, such as 'a', are known as "options" (or, historically, "flags"). Certain options are followed by an "option-argument", as shown with [-c *option\_argument*]. The arguments following the last options and option-arguments are named "operands".
2. Option-arguments are sometimes shown separated from their options by <blank>s, sometimes directly adjacent. This reflects the situation that in some cases an option-argument is included within the same argument string as the option; in most cases it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 203) require that the option be a separate argument from its option-argument, but there are some exceptions in IEEE Std 1003.1-2001 to ensure continued operation of historical applications:
  - a. If the SYNOPSIS of a standard utility shows a <space> between an option and option-argument (as with [-c *option\_argument*] in the example), a conforming application shall use separate arguments for that option and its option-argument.
  - b. If a <space> is not shown (as with [-*foption\_argument*] in the example), a conforming application shall place an option and its option-argument directly adjacent in the same argument string, without intervening <blank>s.
  - c. Notwithstanding the preceding requirements on conforming applications, a conforming implementation shall permit an application to specify options and option-arguments as a single argument or as separate arguments whether or not a <space> is shown on the synopsis line, except in those cases (marked with the XSI portability warning) where an option-argument is optional and no separation can be used.
  - d. A standard utility may also be implemented to operate correctly when the required separation into multiple arguments is violated by a non-conforming application.
3. Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 203) applies. If an option that

does not have option-arguments is repeated, the results are undefined, unless otherwise stated.

4. Frequently, names of parameters that require substitution by actual values are shown with embedded underscores. Alternatively, parameters are shown as follows:

`<parameter name>`

The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and conforming applications shall not include them in data submitted to the utility.

5. When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

`utility_name [-abcDxyz][-p arg][operand]`

Utilities with very complex arguments may be shown as follows:

`utility_name [options][operands]`

6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:

- The number is interpreted as a decimal integer.
- Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
- When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.
- Ranges greater than those listed here are allowed.

This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the IEEE Std 1003.1-2001, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7. Arguments or option-arguments enclosed in the '[' and ']' notation are optional and can be omitted. Conforming applications shall not include the '[' and ']' symbols in data submitted to the utility.
8. Arguments separated by the '|' vertical bar notation are mutually-exclusive. Conforming applications shall not include the '|' symbol in data submitted to the utility. Alternatively, mutually-exclusive options and operands may be listed with multiple synopsis lines. For example:

`utility_name -d[-a][-c option_argument][operand...]`  
`utility_name[-a][-b][operand...]`

When multiple synopsis lines are given for a utility, it is an indication that the utility has mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality of the utility so that only certain other arguments are valid in combination with one of the mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS section, the relationships between arguments depicted in the SYNOPSIS sections are



mandatory requirements placed on conforming applications. The use of conflicting mutually-exclusive arguments produces undefined results, unless a utility description specifies otherwise. When an option is shown without the '[' and ']' brackets, it means that option is required for that version of the SYNOPSIS. However, it is not required to be the first argument, as shown in the example above, unless otherwise stated.

9. Ellipses ("...") are used to denote that one or more occurrences of an option or operand are allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero or more options or operands can be specified. The forms:

```
utility_name -f option_argument...[operand...]
utility_name [-g option_argument]...[operand...]
```

indicate that multiple occurrences of the option and its option-argument preceding the ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See also Guideline 11 in Section 12.2.) In the first example, each option-argument requires a preceding `-f` and at least one `-f option_argument` must be given.

10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities volume of IEEE Std 1003.1-2001, the indented lines following the initial line are continuation lines. An actual use of the command would appear on a single logical line.

## 12.2 Utility Syntax Guidelines

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. The *getopt()* function in the System Interfaces volume of IEEE Std 1003.1-2001 assists utilities in handling options and operands that conform to these guidelines.

Operands and option-arguments can contain characters not specified in the portable character set.

The guidelines are intended to provide guidance to the authors of future utilities, such as those written specific to a local system or that are components of a larger application. Some of the standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections describe the deviations.

**Guideline 1:** Utility names should be between two and nine characters, inclusive.

**Guideline 2:** Utility names should include lowercase letters (the **lower** character classification) and digits only from the portable character set.

**Guideline 3:** Each option name should be a single alphanumeric character (the **alnum** character classification) from the portable character set. The `-W` (capital-W) option shall be reserved for vendor options.

Multi-digit options should not be allowed.

**Guideline 4:** All options should be preceded by the '-' delimiter character.

**Guideline 5:** Options without option-arguments should be accepted when grouped behind one '-' delimiter.

**Guideline 6:** Each option and option-argument should be a separate argument, except as noted in Section 12.1 (on page 201), item (2).

**Guideline 7:** Option-arguments should not be optional.

- 7217       **Guideline 8:**    When multiple option-arguments are specified to follow a single option, they  
 7218                           should be presented as a single argument, using commas within that  
 7219                           argument or <blank>s within that argument to separate them.
- 7220       **Guideline 9:**    All options should precede operands on the command line.
- 7221       **Guideline 10:** The argument -- should be accepted as a delimiter indicating the end of  
 7222                           options. Any following arguments should be treated as operands, even if they  
 7223                           begin with the '-' character. The -- argument should not be used as an  
 7224                           option or as an operand.
- 7225       **Guideline 11:** The order of different options relative to one another should not matter,  
 7226                           unless the options are documented as mutually-exclusive and such an option  
 7227                           is documented to override any incompatible options preceding it. If an option  
 7228                           that has option-arguments is repeated, the option and option-argument  
 7229                           combinations should be interpreted in the order specified on the command  
 7230                           line.
- 7231       **Guideline 12:** The order of operands may matter and position-related interpretations should  
 7232                           be determined on a utility-specific basis.
- 7233       **Guideline 13:** For utilities that use operands to represent files to be opened for either reading  
 7234                           or writing, the '-' operand should be used only to mean standard input (or  
 7235                           standard output when it is clear from context that an output file is being  
 7236                           specified).
- 7237       The utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 that claim conformance to  
 7238       these guidelines shall conform completely to these guidelines as if these guidelines contained the  
 7239       term "shall" instead of "should". On some implementations, the utilities accept usage in  
 7240       violation of these guidelines for backwards-compatibility as well as accepting the required form.
- 7241       It is recommended that all future utilities and applications use these guidelines to enhance user  
 7242       portability. The fact that some historical utilities could not be changed (to avoid breaking  
 7243       existing applications) should not deter this future goal.

# Headers

7244

7245 This chapter describes the contents of headers.

7246 Headers contain function prototypes, the definition of symbolic constants, common structures,  
7247 preprocessor macros, and defined types. Each function in the System Interfaces volume of  
7248 IEEE Std 1003.1-2001 specifies the headers that an application shall include in order to use that  
7249 function. In most cases, only one header is required. These headers are present on an application  
7250 development system; they need not be present on the target execution system.

## 7251 13.1 Format of Entries

7252 The entries in this chapter are based on a common format as follows. The only sections relating  
7253 to conformance are the SYNOPSIS and DESCRIPTION.

### 7254 NAME

7255 This section gives the name or names of the entry and briefly states its purpose.

### 7256 SYNOPSIS

7257 This section summarizes the use of the entry being described.

### 7258 DESCRIPTION

7259 This section describes the functionality of the header.

### 7260 APPLICATION USAGE

7261 This section is informative.

7262 This section gives warnings and advice to application writers about the entry. In the  
7263 event of conflict between warnings and advice and a normative part of this volume of  
7264 IEEE Std 1003.1-2001, the normative material is to be taken as correct.

### 7265 RATIONALE

7266 This section is informative.

7267 This section contains historical information concerning the contents of this volume of  
7268 IEEE Std 1003.1-2001 and why features were included or discarded by the standard  
7269 developers.

### 7270 FUTURE DIRECTIONS

7271 This section is informative.

7272 This section provides comments which should be used as a guide to current thinking;  
7273 there is not necessarily a commitment to adopt these future directions.

### 7274 SEE ALSO

7275 This section is informative.

7276 This section gives references to related information.

### 7277 CHANGE HISTORY

7278 This section is informative.

7279 This section shows the derivation of the entry and any significant changes that have  
7280 been made to it.

## 7281 NAME

7282 aio.h — asynchronous input and output (**REALTIME**)

## 7283 SYNOPSIS

7284 AIO #include &lt;aio.h&gt;

7285

## 7286 DESCRIPTION

7287 The <aio.h> header shall define the **aio\_cb** structure which shall include at least the following  
7288 members:

7289	int	aio_fildes	File descriptor.
7290	off_t	aio_offset	File offset.
7291	volatile void	*aio_buf	Location of buffer.
7292	size_t	aio_nbytes	Length of transfer.
7293	int	aio_reqprio	Request priority offset.
7294	struct sigevent	aio_sigevent	Signal number and value.
7295	int	aio_lio_opcode	Operation to be performed.

7296 This header shall also include the following constants:

7297	AIO_ALLDONE	A return value indicating that none of the requested operations could be
7298		canceled since they are already complete.

7299	AIO_CANCELED	A return value indicating that all requested operations have been
7300		canceled.

7301 AIO\_NOTCANCELED

7302		A return value indicating that some of the requested operations could not
7303		be canceled since they are in progress.

7304	LIO_NOP	A <i>lio_listio()</i> element operation option indicating that no transfer is
7305		requested.

7306	LIO_NOWAIT	A <i>lio_listio()</i> synchronization operation indicating that the calling thread
7307		is to continue execution while the <i>lio_listio()</i> operation is being
7308		performed, and no notification is given when the operation is complete.

7309	LIO_READ	A <i>lio_listio()</i> element operation option requesting a read.
------	----------	---

7310	LIO_WAIT	A <i>lio_listio()</i> synchronization operation indicating that the calling thread
7311		is to suspend until the <i>lio_listio()</i> operation is complete.

7312	LIO_WRITE	A <i>lio_listio()</i> element operation option requesting a write.
------	-----------	--

7313 The following shall be declared as functions and may also be defined as macros. Function  
7314 prototypes shall be provided.

7315	int	aio_cancel(int, struct aiocb *);
7316	int	aio_error(const struct aiocb *);
7317	int	aio_fsync(int, struct aiocb *);
7318	int	aio_read(struct aiocb *);
7319	ssize_t	aio_return(struct aiocb *);
7320	int	aio_suspend(const struct aiocb *const[], int,
7321		const struct timespec *);
7322	int	aio_write(struct aiocb *);
7323	int	lio_listio(int, struct aiocb *restrict const[restrict], int,
7324		struct sigevent *restrict);

7325 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>,  
7326 <signal.h>, <sys/types.h>, and <time.h>.

7327 **APPLICATION USAGE**

7328 None.

7329 **RATIONALE**

7330 None.

7331 **FUTURE DIRECTIONS**

7332 None.

7333 **SEE ALSO**

7334 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of  
7335 IEEE Std 1003.1-2001, *fsync()*, *lseek()*, *read()*, *write()*

7336 **CHANGE HISTORY**

7337 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7338 **Issue 6**

7339 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7340 The description of the constants is expanded.

7341 The **restrict** keyword is added to the prototype for *lio\_listio()*.

7342 **NAME**

7343       arpa/inet.h — definitions for internet operations

7344 **SYNOPSIS**

7345       #include &lt;arpa/inet.h&gt;

7346 **DESCRIPTION**7347       The **in\_port\_t** and **in\_addr\_t** types shall be defined as described in <netinet/in.h>.7348       The **in\_addr** structure shall be defined as described in <netinet/in.h>.7349 IP6       The **INET\_ADDRSTRLEN** and **INET6\_ADDRSTRLEN** macros shall be defined as described in  
7350       <netinet/in.h>.7351       The following shall either be declared as functions, defined as macros, or both. If functions are  
7352       declared, function prototypes shall be provided.

7353       uint32\_t htonl(uint32\_t);

7354       uint16\_t htons(uint16\_t);

7355       uint32\_t ntohl(uint32\_t);

7356       uint16\_t ntohs(uint16\_t);

7357       The **uint32\_t** and **uint16\_t** types shall be defined as described in <inttypes.h>.7358       The following shall be declared as functions and may also be defined as macros. Function  
7359       prototypes shall be provided.

7360       in\_addr\_t     inet\_addr(const char \*);

7361       char         \*inet\_ntoa(struct in\_addr);

7362       const char   \*inet\_ntop(int, const void \*restrict, char \*restrict,  
7363                     socklen\_t);

7364       int          inet\_pton(int, const char \*restrict, void \*restrict);

7365       Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h>  
7366       and <inttypes.h>.7367 **APPLICATION USAGE**

7368       None.

7369 **RATIONALE**

7370       None.

7371 **FUTURE DIRECTIONS**

7372       None.

7373 **SEE ALSO**7374       <netinet/in.h>, <inttypes.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *htonl()*,  
7375       *inet\_addr()*7376 **CHANGE HISTORY**

7377       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7378       The **restrict** keyword is added to the prototypes for *inet\_ntop()* and *inet\_pton()*.

7379 **NAME**

7380       assert.h — verify program assertion

7381 **SYNOPSIS**

7382       #include <assert.h>

7383 **DESCRIPTION**

7384 cx     The functionality described on this reference page is aligned with the ISO C standard. Any  
7385 conflict between the requirements described here and the ISO C standard is unintentional. This  
7386 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7387       The <assert.h> header shall define the *assert()* macro. It refers to the macro NDEBUB which is  
7388 not defined in the header. If NDEBUB is defined as a macro name before the inclusion of this  
7389 header, the *assert()* macro shall be defined simply as:

7390       #define assert(ignore)((void) 0)

7391       Otherwise, the macro behaves as described in *assert()*.

7392       The *assert()* macro shall be redefined according to the current state of NDEBUB each time  
7393 <assert.h> is included.

7394       The *assert()* macro shall be implemented as a macro, not as a function. If the macro definition is  
7395 suppressed in order to access an actual function, the behavior is undefined.

7396 **APPLICATION USAGE**

7397       None.

7398 **RATIONALE**

7399       None.

7400 **FUTURE DIRECTIONS**

7401       None.

7402 **SEE ALSO**

7403       The System Interfaces volume of IEEE Std 1003.1-2001, *assert()*

7404 **CHANGE HISTORY**

7405       First released in Issue 1. Derived from Issue 1 of the SVID.

7406 **Issue 6**

7407       The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999  
7408 standard.

## 7409 NAME

7410 complex.h — complex arithmetic

## 7411 SYNOPSIS

7412 #include &lt;complex.h&gt;

## 7413 DESCRIPTION

7414 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 7415 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7416 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7417 The &lt;complex.h&gt; header shall define the following macros:

7418 complex Expands to **\_Complex**.

7419 \_Complex\_I Expands to a constant expression of type **const float \_Complex**, with the  
 7420 value of the imaginary unit (that is, a number  $i$  such that  $i^2=-1$ ).

7421 imaginary Expands to **\_Imaginary**.

7422 \_Imaginary\_I Expands to a constant expression of type **const float \_Imaginary** with the  
 7423 value of the imaginary unit.

7424 I Expands to either **\_Imaginary\_I** or **\_Complex\_I**. If **\_Imaginary\_I** is not defined,  
 7425 I expands to **\_Complex\_I**.

7426 The macros **imaginary** and **\_Imaginary\_I** shall be defined if and only if the implementation  
 7427 supports imaginary types.

7428 An application may undefine and then, perhaps, redefine the **complex**, **imaginary**, and **I** macros.

7429 The following shall be declared as functions and may also be defined as macros. Function  
 7430 prototypes shall be provided.

7431	double	cabs(double complex);
7432	float	cabsf(float complex);
7433	long double	cabsl(long double complex);
7434	double complex	cacos(double complex);
7435	float complex	cacosf(float complex);
7436	double complex	cacosh(double complex);
7437	float complex	cacoshf(float complex);
7438	long double complex	cacoshl(long double complex);
7439	long double complex	cacosl(long double complex);
7440	double	carg(double complex);
7441	float	cargf(float complex);
7442	long double	cargl(long double complex);
7443	double complex	casin(double complex);
7444	float complex	casinf(float complex);
7445	double complex	casinh(double complex);
7446	float complex	casinhf(float complex);
7447	long double complex	casinhl(long double complex);
7448	long double complex	casinl(long double complex);
7449	double complex	catan(double complex);
7450	float complex	catanf(float complex);
7451	double complex	catanh(double complex);
7452	float complex	catanhf(float complex);
7453	long double complex	catanhl(long double complex);
7454	long double complex	catanl(long double complex);



```

7455     double complex      ccos(double complex);
7456     float complex       ccosf(float complex);
7457     double complex      ccosh(double complex);
7458     float complex       ccoshf(float complex);
7459     long double complex  ccoshl(long double complex);
7460     long double complex  ccosl(long double complex);
7461     double complex      cexp(double complex);
7462     float complex       cexpf(float complex);
7463     long double complex  cexpl(long double complex);
7464     double              cimag(double complex);
7465     float              cimagf(float complex);
7466     long double         cimagl(long double complex);
7467     double complex      clog(double complex);
7468     float complex       clogf(float complex);
7469     long double complex  clogl(long double complex);
7470     double complex      conj(double complex);
7471     float complex       conjf(float complex);
7472     long double complex  conjl(long double complex);
7473     double complex      cpow(double complex, double complex);
7474     float complex       cpowf(float complex, float complex);
7475     long double complex  cpowl(long double complex, long double complex);
7476     double complex      cproj(double complex);
7477     float complex       cprojf(float complex);
7478     long double complex  cprojl(long double complex);
7479     double              creal(double complex);
7480     float              crealf(float complex);
7481     long double         creall(long double complex);
7482     double complex      csin(double complex);
7483     float complex       csinf(float complex);
7484     double complex      csinh(double complex);
7485     float complex       csinhf(float complex);
7486     long double complex  csinhl(long double complex);
7487     long double complex  csinl(long double complex);
7488     double complex      csqrt(double complex);
7489     float complex       csqrtf(float complex);
7490     long double complex  csqrtl(long double complex);
7491     double complex      ctan(double complex);
7492     float complex       ctanf(float complex);
7493     double complex      ctanh(double complex);
7494     float complex       ctanhf(float complex);
7495     long double complex  ctanhl(long double complex);
7496     long double complex  ctanl(long double complex);

```

#### 7497 APPLICATION USAGE

7498 Values are interpreted as radians, not degrees.

#### 7499 RATIONALE

7500 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the  
7501 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the  
7502 imaginary unit by following the inclusion of the <complex.h> header with:

```

7503     #undef I
7504     #define j _Imaginary_I

```

7505 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a  
 7506 sufficiently convenient and more generally useful notation for imaginary terms. The  
 7507 corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or  
 7508 notational convenience will not result in widening types.

7509 On systems with imaginary types, the application has the ability to control whether use of the  
 7510 macro **I** introduces an imaginary type, by explicitly defining **I** to be `_Imaginary_I` or `_Complex_I`.  
 7511 Disallowing imaginary types is useful for some applications intended to run on implementations  
 7512 without support for such types.

7513 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

7514 The *cis()* function ( $\cos(x) + I\sin(x)$ ) was considered but rejected because its implementation is  
 7515 easy and straightforward, even though some implementations could compute sine and cosine  
 7516 more efficiently in tandem.

#### 7517 FUTURE DIRECTIONS

7518 The following function names and the same names suffixed with *f* or *l* are reserved for future  
 7519 use, and may be added to the declarations in the <complex.h> header.

7520	<i>cerf()</i>	<i>cexpm1()</i>	<i>clog2()</i>
7521	<i>cerfc()</i>	<i>clog10()</i>	<i>clgamma()</i>
7522	<i>cexp2()</i>	<i>clog1p()</i>	<i>ctgamma()</i>

#### 7523 SEE ALSO

7524 The System Interfaces volume of IEEE Std 1003.1-2001, *cabs()*, *cacos()*, *cacosh()*, *carg()*, *casin()*,  
 7525 *casinh()*, *catan()*, *catanh()*, *ccos()*, *ccosh()*, *cexp()*, *cimag()*, *clog()*, *conj()*, *cpow()*, *cproj()*, *creal()*,  
 7526 *csin()*, *csinh()*, *csqrt()*, *ctan()*, *ctanh()*

#### 7527 CHANGE HISTORY

7528 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7529 **NAME**

7530 cpio.h — cpio archive values

7531 **SYNOPSIS**

7532 xSI #include <cpio.h>

7533

7534 **DESCRIPTION**

7535 Values needed by the *c\_mode* field of the *cpio* archive format are described as follows:

7536

7537

7538

7539

7540

7541

7542

7543

7544

7545

7546

7547

7548

7549

7550

7551

7552

7553

7554

7555

7556

7557

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

7558 The header shall define the symbolic constant:

7559 MAGIC "070707"

7560 **APPLICATION USAGE**

7561 None.

7562 **RATIONALE**

7563 None.

7564 **FUTURE DIRECTIONS**

7565 None.

7566 **SEE ALSO**

7567 The Shell and Utilities volume of IEEE Std 1003.1-2001, *pax*

7568 **CHANGE HISTORY**

7569 First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988  
7570 standard.

7571 **Issue 6**

7572 The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in the Shell and  
7573 Utilities volume of IEEE Std 1003.1-2001.

7574 **NAME**

7575       ctype.h — character types

7576 **SYNOPSIS**

7577       #include &lt;ctype.h&gt;

7578 **DESCRIPTION**

7579 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 7580       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 7581       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
 7582       symbols in this header.

7583       The following shall be declared as functions and may also be defined as macros. Function  
 7584       prototypes shall be provided.

7585       int    isalnum(int);

7586       int    isalpha(int);

7587 XSI       int    isascii(int);

7588       int    isblank(int);

7589       int    iscntrl(int);

7590       int    isdigit(int);

7591       int    isgraph(int);

7592       int    islower(int);

7593       int    isprint(int);

7594       int    ispunct(int);

7595       int    isspace(int);

7596       int    isupper(int);

7597       int    isxdigit(int);

7598 XSI       int    toascii(int);

7599       int    tolower(int);

7600       int    toupper(int);

7601       The following are defined as macros:

7602 XSI       int    \_toupper(int);

7603       int    \_tolower(int);

7604

7605 **APPLICATION USAGE**

7606       None.

7607 **RATIONALE**

7608       None.

7609 **FUTURE DIRECTIONS**

7610       None.

7611 **SEE ALSO**

7612       <locale.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *isalnum()*, *isalpha()*, *isascii()*,  
 7613       *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*,  
 7614       *mbstowcs()*, *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *\_tolower()*, *toupper()*, *\_toupper()*, *wcstombs()*,  
 7615       *wctomb()*

7616 **CHANGE HISTORY**

7617       First released in Issue 1. Derived from Issue 1 of the SVID.

7618 **Issue 6**

7619 Extensions beyond the ISO C standard are marked.

## 7620 NAME

7621 dirent.h — format of directory entries

## 7622 SYNOPSIS

7623 #include &lt;dirent.h&gt;

## 7624 DESCRIPTION

7625 The internal format of directories is unspecified.

7626 The &lt;dirent.h&gt; header shall define the following type:

7627 **DIR** A type representing a directory stream.7628 It shall also define the structure **dirent** which shall include the following members:7629 XSI `ino_t d_ino` File serial number.7630 `char d_name[]` Name of entry.7631 XSI The type `ino_t` shall be defined as described in <sys/types.h>.7632 The character array `d_name` is of unspecified size, but the number of bytes preceding the terminating null byte shall not exceed {NAME\_MAX}.

7634 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

7636 `int closedir(DIR *);`7637 `DIR *opendir(const char *);`7638 `struct dirent *readdir(DIR *);`7639 TSF `int readdir_r(DIR *restrict, struct dirent *restrict,`  
7640 `struct dirent **restrict);`7641 `void rewinddir(DIR *);`7642 XSI `void seekdir(DIR *, long);`7643 `long telldir(DIR *);`

7644

## 7645 APPLICATION USAGE

7646 None.

## 7647 RATIONALE

7648 Information similar to that in the <dirent.h> header is contained in a file <sys/dir.h> in 4.2 BSD  
 7649 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of  
 7650 IEEE Std 1003.1-2001 is **struct direct**. The filename was changed because the name <sys/dir.h>  
 7651 was also used in earlier implementations to refer to definitions related to the older access  
 7652 method; this produced name conflicts. The name of the structure was changed because this  
 7653 volume of IEEE Std 1003.1-2001 does not completely define what is in the structure, so it could  
 7654 be different on some implementations from **struct direct**.

7655 The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:7656 `sizeof(d_name)`

7657 is incorrect; use:

7658 `strlen(d_name)`

7659 instead.

7660 The array of **char** `d_name` is not a fixed size. Implementations may need to declare **struct dirent**  
 7661 with an array size for `d_name` of 1, but the actual number of characters provided matches (or  
 7662 only slightly exceeds) the length of the filename.

7663 **FUTURE DIRECTIONS**

7664       None.

7665 **SEE ALSO**

7666       <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *closedir()*, *opendir()*,  
7667       *readdir()*, *readdir\_r()*, *rewinddir()*, *seekdir()*, *telldir()*

7668 **CHANGE HISTORY**

7669       First released in Issue 2.

7670 **Issue 5**

7671       The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7672 **Issue 6**7673       The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir\_r()*.7674       The **restrict** keyword is added to the prototype for *readdir\_r()*.

7675 **NAME**

7676       dlfcn.h — dynamic linking

7677 **SYNOPSIS**

7678 XSI     #include &lt;dlfcn.h&gt;

7679

7680 **DESCRIPTION**7681       The <dlfcn.h> header shall define at least the following macros for use in the construction of a  
7682       *dlopen()* *mode* argument:

7683       RTLD\_LAZY       Relocations are performed at an implementation-defined time.

7684       RTLD\_NOW        Relocations are performed when the object is loaded.

7685       RTLD\_GLOBAL     All symbols are available for relocation processing of other modules.

7686       RTLD\_LOCAL     All symbols are not made available for relocation processing by other  
7687       modules.7688       The following shall be declared as functions and may also be defined as macros. Function  
7689       prototypes shall be provided.

7690       int     dlclose(void \*);

7691       char    \*dlerror(void);

7692       void    \*dlopen(const char \*, int);

7693       void    \*dlsym(void \*restrict, const char \*restrict);

7694 **APPLICATION USAGE**

7695       None.

7696 **RATIONALE**

7697       None.

7698 **FUTURE DIRECTIONS**

7699       None.

7700 **SEE ALSO**7701       The System Interfaces volume of IEEE Std 1003.1-2001, *dlopen()*, *dlclose()*, *dlsym()*, *dlerror()*7702 **CHANGE HISTORY**

7703       First released in Issue 5.

7704 **Issue 6**7705       The **restrict** keyword is added to the prototype for *dlsym()*.



7706 **NAME**

7707       errno.h — system error numbers

7708 **SYNOPSIS**

7709       #include <errno.h>

7710 **DESCRIPTION**

7711 **CX**       Some of the functionality described on this reference page extends the ISO C standard. Any  
7712 conflict between the requirements described here and the ISO C standard is unintentional. This  
7713 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7714 **CX**       The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7715       The <errno.h> header shall provide a declaration for *errno* and give positive values for the  
7716 following symbolic constants. Their values shall be unique except as noted below.

7717	[E2BIG]	Argument list too long.
7718	[EACCES]	Permission denied.
7719	[EADDRINUSE]	Address in use.
7720	[EADDRNOTAVAIL]	Address not available.
7721	[EAFNOSUPPORT]	Address family not supported.
7722	[EAGAIN]	Resource unavailable, try again (may be the same value as
7723		[EWOULDBLOCK]).
7724	[EALREADY]	Connection already in progress.
7725	[EBADF]	Bad file descriptor.
7726	[EBADMSG]	Bad message.
7727	[EBUSY]	Device or resource busy.
7728	[ECANCELED]	Operation canceled.
7729	[ECHILD]	No child processes.
7730	[ECONNABORTED]	Connection aborted.
7731	[ECONNREFUSED]	Connection refused.
7732	[ECONNRESET]	Connection reset.
7733	[EDEADLK]	Resource deadlock would occur.
7734	[EDESTADDRREQ]	Destination address required.
7735	[EDOM]	Mathematics argument out of domain of function.
7736	[EDQUOT]	Reserved.
7737	[EEXIST]	File exists.
7738	[EFAULT]	Bad address.
7739	[EFBIG]	File too large.
7740	[EHOSTUNREACH]	Host is unreachable.
7741	[EIDRM]	Identifier removed.
7742	[EILSEQ]	Illegal byte sequence.

7743		[EINPROGRESS]	Operation in progress.
7744		[EINTR]	Interrupted function.
7745		[EINVAL]	Invalid argument.
7746		[EIO]	I/O error.
7747		[EISCONN]	Socket is connected.
7748		[EISDIR]	Is a directory.
7749		[ELOOP]	Too many levels of symbolic links.
7750		[EMFILE]	Too many open files.
7751		[EMLINK]	Too many links.
7752		[EMSGSIZE]	Message too large.
7753		[EMULTIHOP]	Reserved.
7754		[ENAMETOOLONG]	Filename too long.
7755		[ENETDOWN]	Network is down.
7756		[ENETRESET]	Connection aborted by network.
7757		[ENETUNREACH]	Network unreachable.
7758		[ENFILE]	Too many files open in system.
7759		[ENOBUFS]	No buffer space available.
7760	XSR	[ENODATA]	No message is available on the STREAM head read queue.
7761		[ENODEV]	No such device.
7762		[ENOENT]	No such file or directory.
7763		[ENOEXEC]	Executable file format error.
7764		[ENOLCK]	No locks available.
7765		[ENOLINK]	Reserved.
7766		[ENOMEM]	Not enough space.
7767		[ENOMSG]	No message of the desired type.
7768		[ENOPROTOPT]	Protocol not available.
7769		[ENOSPC]	No space left on device.
7770	XSR	[ENOSR]	No STREAM resources.
7771	XSR	[ENOSTR]	Not a STREAM.
7772		[ENOSYS]	Function not supported.
7773		[ENOTCONN]	The socket is not connected.
7774		[ENOTDIR]	Not a directory.
7775		[ENOTEMPTY]	Directory not empty.
7776		[ENOTSOCK]	Not a socket.

7777	[ENOTSUP]	Not supported.
7778	[ENOTTY]	Inappropriate I/O control operation.
7779	[ENXIO]	No such device or address.
7780	[EOPNOTSUPP]	Operation not supported on socket.
7781	[EOVERFLOW]	Value too large to be stored in data type.
7782	[EPERM]	Operation not permitted.
7783	[EPIPE]	Broken pipe.
7784	[EPROTO]	Protocol error.
7785	[EPROTONOSUPPORT]	
7786		Protocol not supported.
7787	[EPROTOTYPE]	Protocol wrong type for socket.
7788	[ERANGE]	Result too large.
7789	[EROFS]	Read-only file system.
7790	[ESPIPE]	Invalid seek.
7791	[ESRCH]	No such process.
7792	[ESTALE]	Reserved.
7793	XSR [ETIME]	Stream <i>ioctl()</i> timeout.
7794	[ETIMEDOUT]	Connection timed out.
7795	[ETXTBSY]	Text file busy.
7796	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7797	[EXDEV]	Cross-device link.

## 7798 APPLICATION USAGE

7799 Additional error numbers may be defined on conforming systems; see the System Interfaces  
7800 volume of IEEE Std 1003.1-2001.

## 7801 RATIONALE

7802 None.

## 7803 FUTURE DIRECTIONS

7804 None.

## 7805 SEE ALSO

7806 The System Interfaces volume of IEEE Std 1003.1-2001, Section 2.3, Error Numbers

## 7807 CHANGE HISTORY

7808 First released in Issue 1. Derived from Issue 1 of the SVID.

## 7809 Issue 5

7810 Updated for alignment with the POSIX Realtime Extension.

## 7811 Issue 6

7812 The following new requirements on POSIX implementations derive from alignment with the  
7813 Single UNIX Specification:

- 7814 • The majority of the error conditions previously marked as extensions are now mandatory,  
7815 except for the STREAMS-related error conditions.

7816 Values for *errno* are now required to be distinct positive values rather than non-zero values. This  
7817 change is for alignment with the ISO/IEC 9899:1999 standard.

7818 **NAME**

7819       fcntl.h — file control options

7820 **SYNOPSIS**

7821       #include <fcntl.h>

7822 **DESCRIPTION**

7823       The <fcntl.h> header shall define the following requests and arguments for use by the functions  
7824       *fcntl()* and *open()*.

7825       Values for *cmd* used by *fcntl()* (the following values are unique) are as follows:

- 7826       F\_DUPFD       Duplicate file descriptor.
- 7827       F\_GETFD       Get file descriptor flags.
- 7828       F\_SETFD       Set file descriptor flags.
- 7829       F\_GETFL       Get file status flags and file access modes.
- 7830       F\_SETFL       Set file status flags.
- 7831       F\_GETLK       Get record locking information.
- 7832       F\_SETLK       Set record locking information.
- 7833       F\_SETLKW      Set record locking information; wait if blocked.
- 7834       F\_GETOWN      Get process or process group ID to receive SIGURG signals.
- 7835       F\_SETOWN      Set process or process group ID to receive SIGURG signals.

7836       File descriptor flags used for *fcntl()* are as follows:

- 7837       FD\_CLOEXEC    Close the file descriptor upon execution of an *exec* family function.

7838       Values for *l\_type* used for record locking with *fcntl()* (the following values are unique) are as  
7839       follows:

- 7840       F\_RDLCK       Shared or read lock.
- 7841       F\_UNLCK       Unlock.
- 7842       F\_WRLCK       Exclusive or write lock.

7843 XSI       The values used for *l\_whence*, *SEEK\_SET*, *SEEK\_CUR*, and *SEEK\_END* shall be defined as  
7844       described in <unistd.h>.

7845       The following values are file creation flags and are used in the *oflag* value to *open()*. They shall  
7846       be bitwise-distinct.

- 7847       O\_CREAT        Create file if it does not exist.
- 7848       O\_EXCL        Exclusive use flag.
- 7849       O\_NOCTTY       Do not assign controlling terminal.
- 7850       O\_TRUNC        Truncate flag.

7851       File status flags used for *open()* and *fcntl()* are as follows:

- 7852       O\_APPEND       Set append mode.
- 7853 SIO       O\_DSYNC    Write according to synchronized I/O data integrity completion.
- 7854       O\_NONBLOCK    Non-blocking mode.

7855	SIO	<b>O_RSYNC</b>	Synchronized read I/O operations.
7856		<b>O_SYNC</b>	Write according to synchronized I/O file integrity completion.
7857		Mask for use with file access modes is as follows:	
7858		<b>O_ACCMODE</b>	Mask for file access modes.
7859		File access modes used for <i>open()</i> and <i>fcntl()</i> are as follows:	
7860		<b>O_RDONLY</b>	Open for reading only.
7861		<b>O_RDWR</b>	Open for reading and writing.
7862		<b>O_WRONLY</b>	Open for writing only.
7863	XSI	The symbolic names for file modes for use as values of <b>mode_t</b> shall be defined as described in	
7864		<sys/stat.h>.	
7865	ADV	Values for <i>advice</i> used by <i>posix_fadvise()</i> are as follows:	
7866		<b>POSIX_FADV_NORMAL</b>	
7867		The application has no advice to give on its behavior with respect to the specified data. It is	
7868		the default characteristic if no advice is given for an open file.	
7869		<b>POSIX_FADV_SEQUENTIAL</b>	
7870		The application expects to access the specified data sequentially from lower offsets to	
7871		higher offsets.	
7872		<b>POSIX_FADV_RANDOM</b>	
7873		The application expects to access the specified data in a random order.	
7874		<b>POSIX_FADV_WILLNEED</b>	
7875		The application expects to access the specified data in the near future.	
7876		<b>POSIX_FADV_DONTNEED</b>	
7877		The application expects that it will not access the specified data in the near future.	
7878		<b>POSIX_FADV_NOREUSE</b>	
7879		The application expects to access the specified data once and then not reuse it thereafter.	
7880			
7881		The structure <b> flock</b> describes a file lock. It shall include the following members:	
7882		short l_type	Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.
7883		short l_whence	Flag for starting offset.
7884		off_t l_start	Relative offset in bytes.
7885		off_t l_len	Size; if 0 then until EOF.
7886		pid_t l_pid	Process ID of the process holding the lock; returned with F_GETLK.
7887		The <b>mode_t</b> , <b>off_t</b> , and <b>pid_t</b> types shall be defined as described in <sys/types.h>.	
7888		The following shall be declared as functions and may also be defined as macros. Function	
7889		prototypes shall be provided.	
7890		int creat(const char *, mode_t);	
7891		int fcntl(int, int, ...);	
7892		int open(const char *, int, ...);	
7893	ADV	int posix_fadvise(int, off_t, off_t, int);	
7894		int posix_fallocate(int, off_t, off_t);	
7895			

2  
2

7896 XSI Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and  
7897 <unistd.h>.

## 7898 APPLICATION USAGE

7899 None.

## 7900 RATIONALE

7901 None.

## 7902 FUTURE DIRECTIONS

7903 None.

## 7904 SEE ALSO

7905 <sys/stat.h>, <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std 1003.1-2001,  
7906 *creat()*, *exec*, *fcntl()*, *open()*, *posix\_fadvise()*, *posix\_fallocate()*, *posix\_madvise()*

## 7907 CHANGE HISTORY

7908 First released in Issue 1. Derived from Issue 1 of the SVID.

### 7909 Issue 5

7910 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

### 7911 Issue 6

7912 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 7913 • O\_DSYNC and O\_RSYNC are marked as part of the Synchronized Input and Output option.

7914 The following new requirements on POSIX implementations derive from alignment with the  
7915 Single UNIX Specification:

- 7916 • The definition of the **mode\_t**, **off\_t**, and **pid\_t** types is mandated.

7917 The F\_GETOWN and F\_SETOWN values are added for sockets.

7918 The *posix\_fadvise()*, *posix\_fallocate()*, and *posix\_madvise()* functions are added for alignment with  
7919 IEEE Std 1003.1d-1999.

7920 IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix\_madvise()* to  
7921 <sys/mman.h>.

7922 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/18 is applied, updating the prototypes for 2  
7923 *posix\_fadvise()* and *posix\_fallocate()* to be large file-aware, using **off\_t** instead of **size\_t**. 2

## 7924 NAME

7925 fenv.h — floating-point environment

## 7926 SYNOPSIS

7927 #include &lt;fenv.h&gt;

## 7928 DESCRIPTION

7929 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 7930 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7931 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7932 The <fenv.h> header shall define the following data types through **typedef**:

7933 **fenv\_t** Represents the entire floating-point environment. The floating-point environment  
 7934 refers collectively to any floating-point status flags and control modes supported  
 7935 by the implementation.

7936 **fexcept\_t** Represents the floating-point status flags collectively, including any status the  
 7937 implementation associates with the flags. A floating-point status flag is a system  
 7938 variable whose value is set (but never cleared) when a floating-point exception is  
 7939 raised, which occurs as a side effect of exceptional floating-point arithmetic to  
 7940 provide auxiliary information. A floating-point control mode is a system variable  
 7941 whose value may be set by the user to affect the subsequent behavior of floating-  
 7942 point arithmetic.

7943 The <fenv.h> header shall define the following constants if and only if the implementation  
 7944 supports the floating-point exception by means of the floating-point functions *feclearexcept()*,  
 7945 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. Each expands to an integer  
 7946 constant expression with values such that bitwise-inclusive ORs of all combinations of the  
 7947 constants result in distinct values.

7948 FE\_DIVBYZERO  
 7949 FE\_INEXACT  
 7950 FE\_INVALID  
 7951 FE\_OVERFLOW  
 7952 FE\_UNDERFLOW

7953 The <fenv.h> header shall define the following constant, which is simply the bitwise-inclusive  
 7954 OR of all floating-point exception constants defined above:

7955 FE\_ALL\_EXCEPT

7956 The <fenv.h> header shall define the following constants if and only if the implementation  
 7957 supports getting and setting the represented rounding direction by means of the *fegetround()*  
 7958 and *fesetround()* functions. Each expands to an integer constant expression whose values are  
 7959 distinct non-negative values.

7960 FE\_DOWNWARD  
 7961 FE\_TONEAREST  
 7962 FE\_TOWARDZERO  
 7963 FE\_UPWARD

7964 The <fenv.h> header shall define the following constant, which represents the default floating-  
 7965 point environment (that is, the one installed at program startup) and has type pointer to const-  
 7966 qualified **fenv\_t**. It can be used as an argument to the functions within the <fenv.h> header that  
 7967 manage the floating-point environment.

7968 FE\_DFL\_ENV



7969 The following shall be declared as functions and may also be defined as macros. Function  
7970 prototypes shall be provided.

```
7971 int feclearexcept(int);
7972 int fegetexceptflag(fexcept_t *, int);
7973 int feraiseexcept(int);
7974 int fesetexceptflag(const fexcept_t *, int);
7975 int fetestexcept(int);
7976 int fegetround(void);
7977 int fesetround(int);
7978 int fegetenv(fenv_t *);
7979 int feholdexcept(fenv_t *);
7980 int fesetenv(const fenv_t *);
7981 int feupdateenv(const fenv_t *);
```

7982 The FENV\_ACCESS pragma provides a means to inform the implementation when an  
7983 application might access the floating-point environment to test floating-point status flags or run  
7984 under non-default floating-point control modes. The pragma shall occur either outside external  
7985 declarations or preceding all explicit declarations and statements inside a compound statement.  
7986 When outside external declarations, the pragma takes effect from its occurrence until another  
7987 FENV\_ACCESS pragma is encountered, or until the end of the translation unit. When inside a  
7988 compound statement, the pragma takes effect from its occurrence until another FENV\_ACCESS  
7989 pragma is encountered (including within a nested compound statement), or until the end of the  
7990 compound statement; at the end of a compound statement the state for the pragma is restored to  
7991 its condition just before the compound statement. If this pragma is used in any other context, the  
7992 behavior is undefined. If part of an application tests floating-point status flags, sets floating-  
7993 point control modes, or runs under non-default mode settings, but was translated with the state  
7994 for the FENV\_ACCESS pragma off, the behavior is undefined. The default state (on or off) for  
7995 the pragma is implementation-defined. (When execution passes from a part of the application  
7996 translated with FENV\_ACCESS off to a part translated with FENV\_ACCESS on, the state of the  
7997 floating-point status flags is unspecified and the floating-point control modes have their default  
7998 settings.)

## 7999 APPLICATION USAGE

8000 This header is designed to support the floating-point exception status flags and directed-  
8001 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-  
8002 point state information. Also it is designed to facilitate code portability among all systems.

8003 Certain application programming conventions support the intended model of use for the  
8004 floating-point environment:

- 8005 • A function call does not alter its caller's floating-point control modes, clear its caller's  
8006 floating-point status flags, nor depend on the state of its caller's floating-point status flags  
8007 unless the function is so documented.
- 8008 • A function call is assumed to require default floating-point control modes, unless its  
8009 documentation promises otherwise.
- 8010 • A function call is assumed to have the potential for raising floating-point exceptions, unless  
8011 its documentation promises otherwise.

8012 With these conventions, an application can safely assume default floating-point control modes  
8013 (or be unaware of them). The responsibilities associated with accessing the floating-point  
8014 environment fall on the application that does so explicitly.

8015 Even though the rounding direction macros may expand to constants corresponding to the  
8016 values of FLT\_ROUNDS, they are not required to do so.

8017 For example:

```
8018 #include <fenv.h>
8019 void f(double x)
8020 {
8021     #pragma STDC FENV_ACCESS ON
8022     void g(double);
8023     void h(double);
8024     /* ... */
8025     g(x + 1);
8026     h(x + 1);
8027     /* ... */
8028 }
```

8029 If the function *g()* might depend on status flags set as a side effect of the first *x+1*, or if the  
8030 second *x+1* might depend on control modes set as a side effect of the call to function *g()*, then  
8031 the application shall contain an appropriately placed invocation as follows:

```
8032 #pragma STDC FENV_ACCESS ON
```

### 8033 RATIONALE

#### 8034 The **fexcept\_t** Type

8035 **fexcept\_t** does not have to be an integer type. Its values must be obtained by a call to  
8036 *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An  
8037 implementation might simply implement **fexcept\_t** as an **int** and use the representations  
8038 reflected by the exception macros, but is not required to; other representations might contain  
8039 extra information about the exceptions. **fexcept\_t** might be a **struct** with a member for each  
8040 exception (that might hold the address of the first or last floating-point instruction that caused  
8041 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an  
8042 **fexcept\_t**, and so the user cannot inspect it.

#### 8043 Exception and Rounding Macros

8044 Macros corresponding to unsupported modes and rounding directions are not defined by the  
8045 implementation and must not be defined by the application. An application might use **#ifdef** to  
8046 test for this.

### 8047 FUTURE DIRECTIONS

8048 None.

### 8049 SEE ALSO

8050 The System Interfaces volume of IEEE Std 1003.1-2001, *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*,  
8051 *fegetround()*, *fehldexcept()*, *feraiseexcept()*, *fesetenv()*, *fesetexceptflag()*, *fesetround()*, *fetestexcept()*,  
8052 *feupdateenv()*

### 8053 CHANGE HISTORY

8054 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8055 The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*,  
8056 *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the  
8057 ISO/IEC 9899:1999 standard, Defect Report 202.

8058 **NAME**

8059 float.h — floating types

8060 **SYNOPSIS**

8061 #include <float.h>

8062 **DESCRIPTION**

8063 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 8064 conflict between the requirements described here and the ISO C standard is unintentional. This  
 8065 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8066 The characteristics of floating types are defined in terms of a model that describes a  
 8067 representation of floating-point numbers and values that provide information about an  
 8068 implementation's floating-point arithmetic.

8069 The following parameters are used to define the model for each floating-point type:

8070 *s* Sign ( $\pm 1$ ).

8071 *b* Base or radix of exponent representation (an integer  $> 1$ ).

8072 *e* Exponent (an integer between a minimum  $e_{\min}$  and a maximum  $e_{\max}$ ).

8073 *p* Precision (the number of base-*b* digits in the significand).

8074  $f_k$  Non-negative integers less than *b* (the significand digits).

8075 A floating-point number *x* is defined by the following model:

$$8076 \quad x = sb^e \sum_{k=1}^p f_k b^{-k}, \quad e_{\min} \leq e \leq e_{\max}$$

8077 In addition to normalized floating-point numbers ( $f_1 > 0$  if  $x \neq 0$ ), floating types may be able to  
 8078 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ( $x \neq 0$ ,  
 8079  $e = e_{\min}$ ,  $f_1 = 0$ ) and unnormalized floating-point numbers ( $x \neq 0$ ,  $e > e_{\min}$ ,  $f_1 = 0$ ), and values that are  
 8080 not floating-point numbers, such as infinities and NaNs. A NaN is an encoding signifying Not-  
 8081 a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a  
 8082 floating-point exception; a *signaling NaN* generally raises a floating-point exception when  
 8083 occurring as an arithmetic operand.

8084 The accuracy of the floating-point operations ( $' + '$ ,  $' - '$ ,  $' * '$ ,  $' / '$ ) and of the library functions  
 8085 in <math.h> and <complex.h> that return floating-point results is implementation-defined. The  
 8086 implementation may state that the accuracy is unknown.

8087 All integer values in the <float.h> header, except FLT\_ROUNDS, shall be constant expressions  
 8088 suitable for use in #if preprocessing directives; all floating values shall be constant expressions.  
 8089 All except DECIMAL\_DIG, FLT\_EVAL\_METHOD, FLT\_RADIX, and FLT\_ROUNDS have  
 8090 separate names for all three floating-point types. The floating-point model representation is  
 8091 provided for all values except FLT\_EVAL\_METHOD and FLT\_ROUNDS.

8092 The rounding mode for floating-point addition is characterized by the implementation-defined  
 8093 value of FLT\_ROUNDS:

8094 -1 Indeterminable.

8095 0 Toward zero.

8096 1 To nearest.

8097 2 Toward positive infinity.

8098           3   Toward negative infinity.

8099       All other values for FLT\_ROUNDS characterize implementation-defined rounding behavior.

8100       The values of operations with floating operands and values subject to the usual arithmetic  
8101 conversions and of floating constants are evaluated to a format whose range and precision may  
8102 be greater than required by the type. The use of evaluation formats is characterized by the  
8103 implementation-defined value of FLT\_EVAL\_METHOD:

8104       -1   Indeterminable.

8105       0   Evaluate all operations and constants just to the range and precision of the type.

8106       1   Evaluate operations and constants of type **float** and **double** to the range and precision of the  
8107 **double** type; evaluate **long double** operations and constants to the range and precision of  
8108 the **long double** type.

8109       2   Evaluate all operations and constants to the range and precision of the **long double** type.

8110       All other negative values for FLT\_EVAL\_METHOD characterize implementation-defined  
8111 behavior.

8112       The values given in the following list shall be defined as constant expressions with  
8113 implementation-defined values that are greater or equal in magnitude (absolute value) to those  
8114 shown, with the same sign.

8115       • Radix of exponent representation,  $b$ .

8116       FLT\_RADIX           2

8117       • Number of base-FLT\_RADIX digits in the floating-point significand,  $p$ .

8118       FLT\_MANT\_DIG

8119       DBL\_MANT\_DIG

8120       LDBL\_MANT\_DIG

8121       • Number of decimal digits,  $n$ , such that any floating-point number in the widest supported  
8122 floating type with  $p_{\max}$  radix  $b$  digits can be rounded to a floating-point number with  $n$   
8123 decimal digits and back again without change to the value.

8124       
$$\begin{cases} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \text{otherwise} \end{cases}$$

8125       DECIMAL\_DIG       10

8126       • Number of decimal digits,  $q$ , such that any floating-point number with  $q$  decimal digits can  
8127 be rounded into a floating-point number with  $p$  radix  $b$  digits and back again without change  
8128 to the  $q$  decimal digits.

8129       
$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil (p - 1) \log_{10} b \right\rceil & \text{otherwise} \end{cases}$$

8130       FLT\_DIG           6

8131       DBL\_DIG           10

8132	LDBL_DIG	10
8133	• Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a	
8134	normalized floating-point number, $e_{\min}$ .	
8135	FLT_MIN_EXP	
8136	DBL_MIN_EXP	
8137	LDBL_MIN_EXP	
8138	• Minimum negative integer such that 10 raised to that power is in the range of normalized	
8139	floating-point numbers.	
8140	$\left\lceil \log_{10} b^{e_{\min} - 1} \right\rceil$	
8141	FLT_MIN_10_EXP	−37
8142	DBL_MIN_10_EXP	−37
8143	LDBL_MIN_10_EXP	−37
8144	• Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable	
8145	finite floating-point number, $e_{\max}$ .	
8146	FLT_MAX_EXP	
8147	DBL_MAX_EXP	
8148	LDBL_MAX_EXP	
8149	• Maximum integer such that 10 raised to that power is in the range of representable finite	
8150	floating-point numbers.	
8151	$\left\lceil \log_{10} ((1 - b^{-p}) b^{e_{\max}}) \right\rceil$	
8152	FLT_MAX_10_EXP	+37
8153	DBL_MAX_10_EXP	+37
8154	LDBL_MAX_10_EXP	+37
8155	The values given in the following list shall be defined as constant expressions with	
8156	implementation-defined values that are greater than or equal to those shown:	
8157	• Maximum representable finite floating-point number.	
8158	$(1 - b^{-p}) b^{e_{\max}}$	
8159	FLT_MAX	1E+37
8160	DBL_MAX	1E+37
8161	LDBL_MAX	1E+37
8162	The values given in the following list shall be defined as constant expressions with	
8163	implementation-defined (positive) values that are less than or equal to those shown:	
8164	• The difference between 1 and the least value greater than 1 that is representable in the given	
8165	floating-point type, $b^{1-p}$ .	
8166	FLT_EPSILON	1E−5
8167	DBL_EPSILON	1E−9

8168           LDBL\_EPSILON       1E-9

8169           • Minimum normalized positive floating-point number,  $b^{e_{\min}-1}$ .

8170           FLT\_MIN           1E-37

8171           DBL\_MIN           1E-37

8172           LDBL\_MIN          1E-37

8173 **APPLICATION USAGE**

8174           None.

8175 **RATIONALE**

8176           None.

8177 **FUTURE DIRECTIONS**

8178           None.

8179 **SEE ALSO**

8180           <complex.h>, <math.h>

8181 **CHANGE HISTORY**

8182           First released in Issue 4. Derived from the ISO C standard.

8183 **Issue 6**

8184           The description of the operations with floating-point values is updated for alignment with the  
8185           ISO/IEC 9899:1999 standard.

# 8186 NAME

8187       fmtmsg.h — message display structures

# 8188 SYNOPSIS

8189 XSI       #include <fmtmsg.h>

# 8191 DESCRIPTION

8192       The <fmtmsg.h> header shall define the following macros, which expand to constant integer expressions:

8194	MM_HARD	Source of the condition is hardware.
8195	MM_SOFT	Source of the condition is software.
8196	MM_FIRM	Source of the condition is firmware.
8197	MM_APPL	Condition detected by application.
8198	MM_UTIL	Condition detected by utility.
8199	MM_OPSYS	Condition detected by operating system.
8200	MM_RECOVER	Recoverable error.
8201	MM_NRECOV	Non-recoverable error.
8202	MM_HALT	Error causing application to halt.
8203	MM_ERROR	Application has encountered a non-fatal fault.
8204	MM_WARNING	Application has detected unusual non-error condition.
8205	MM_INFO	Informative message.
8206	MM_NOSEV	No severity level provided for the message.
8207	MM_PRINT	Display message on standard error.
8208	MM_CONSOLE	Display message on system console.

8209       The table below indicates the null values and identifiers for *fmtmsg()* arguments. The <fmtmsg.h> header shall define the macros in the **Identifier** column, which expand to constant expressions that expand to expressions of the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
8214 <i>label</i>	char *	(char*)0	MM_NULLLBL
8215 <i>severity</i>	int	0	MM_NULLSEV
8216 <i>class</i>	long	0L	MM_NULLMC
8217 <i>text</i>	char *	(char*)0	MM_NULLTXT
8218 <i>action</i>	char *	(char*)0	MM_NULLACT
8219 <i>tag</i>	char *	(char*)0	MM_NULLTAG

8220       The <fmtmsg.h> header shall also define the following macros for use as return values for *fmtmsg()*:

8222	MM_OK	The function succeeded.
8223	MM_NOTOK	The function failed completely.
8224	MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.

8226           MM\_NOCON           The function was unable to generate a console message, but otherwise  
8227                               succeeded.

8228           The following shall be declared as a function and may also be defined as a macro. A function  
8229           prototype shall be provided.

8230           int fmtmsg(long, const char \*, int,  
8231               const char \*, const char \*, const char \*);

8232   **APPLICATION USAGE**

8233           None.

8234   **RATIONALE**

8235           None.

8236   **FUTURE DIRECTIONS**

8237           None.

8238   **SEE ALSO**

8239           The System Interfaces volume of IEEE Std 1003.1-2001, *fmtmsg()*

8240   **CHANGE HISTORY**

8241           First released in Issue 4, Version 2.



8242 **NAME**

8243       fnmatch.h — filename-matching types

8244 **SYNOPSIS**

8245       #include <fnmatch.h>

8246 **DESCRIPTION**

8247       The <fnmatch.h> header shall define the following constants:

8248       FNM\_NOMATCH     The string does not match the specified pattern.

8249       FNM\_PATHNAME    Slash in *string* only matches slash in *pattern*.

8250       FNM\_PERIOD       Leading period in *string* must be exactly matched by period in *pattern*.

8251       FNM\_NOESCAPE    Disable backslash escaping.

8252   OB XSI   FNM\_NOSYS     Reserved.

8253       The following shall be declared as a function and may also be defined as a macro. A function  
8254       prototype shall be provided.

8255       int fnmatch(const char \*, const char \*, int);

8256 **APPLICATION USAGE**

8257       None.

8258 **RATIONALE**

8259       None.

8260 **FUTURE DIRECTIONS**

8261       None.

8262 **SEE ALSO**

8263       The System Interfaces volume of IEEE Std 1003.1-2001, *fnmatch()*, the Shell and Utilities volume  
8264       of IEEE Std 1003.1-2001

8265 **CHANGE HISTORY**

8266       First released in Issue 4. Derived from the ISO POSIX-2 standard.

8267 **Issue 6**

8268       The constant FNM\_NOSYS is marked obsolescent.

8269 **NAME**

8270 ftw.h — file tree traversal

8271 **SYNOPSIS**8272 XSI `#include <ftw.h>`

8273

8274 **DESCRIPTION**8275 The <ftw.h> header shall define the **FTW** structure that includes at least the following members:8276 `int base`8277 `int level`8278 The <ftw.h> header shall define macros for use as values of the third argument to the  
8279 application-supplied function that is passed as the second argument to *ftw()* and *nftw()*:8280 `FTW_F` File.8281 `FTW_D` Directory.8282 `FTW_DNR` Directory without read permission.8283 `FTW_DP` Directory with subdirectories visited.8284 `FTW_NS` Unknown type; *stat()* failed.8285 `FTW_SL` Symbolic link.8286 `FTW_SLN` Symbolic link that names a nonexistent file.8287 The <ftw.h> header shall define macros for use as values of the fourth argument to *nftw()*:8288 `FTW_PHYS` Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows  
8289 links but does not walk down any path that crosses itself.8290 `FTW_MOUNT` The walk does not cross a mount point.8291 `FTW_DEPTH` All subdirectories are visited before the directory itself.8292 `FTW_CHDIR` The walk changes to each directory before reading it.8293 The following shall be declared as functions and may also be defined as macros. Function  
8294 prototypes shall be provided.8295 `int ftw(const char *, int (*)(const char *, const struct stat *,`  
8296 `int), int);`8297 `int nftw(const char *, int (*)(const char *, const struct stat *,`  
8298 `int, struct FTW*), int, int);`8299 The <ftw.h> header shall define the **stat** structure and the symbolic names for *st\_mode* and the  
8300 file type test macros as described in <sys/stat.h>.

8301 Inclusion of the &lt;ftw.h&gt; header may also make visible all symbols from &lt;sys/stat.h&gt;.

8302 **APPLICATION USAGE**

8303       None.

8304 **RATIONALE**

8305       None.

8306 **FUTURE DIRECTIONS**

8307       None.

8308 **SEE ALSO**

8309       <sys/stat.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *ftw()*, *nftw()*

8310 **CHANGE HISTORY**

8311       First released in Issue 1. Derived from Issue 1 of the SVID.

8312 **Issue 5**

8313       A description of FTW\_DP is added.

## 8314 NAME

8315 glob.h — pathname pattern-matching types

## 8316 SYNOPSIS

8317 #include &lt;glob.h&gt;

## 8318 DESCRIPTION

8319 The <glob.h> header shall define the structures and symbolic constants used by the *glob()*  
8320 function.8321 The structure type **glob\_t** shall contain at least the following members:8322 size\_t gl\_pathc Count of paths matched by *pattern*.

8323 char \*\*gl\_pathv Pointer to a list of matched pathnames.

8324 size\_t gl\_offs Slots to reserve at the beginning of *gl\_pathv*.8325 The following constants shall be provided as values for the *flags* argument:

8326 GLOB\_APPEND Append generated pathnames to those previously obtained.

8327 GLOB\_DOOFFS Specify how many null pointers to add to the beginning of *gl\_pathv*.8328 GLOB\_ERR Cause *glob()* to return on error.8329 GLOB\_MARK Each pathname that is a directory that matches *pattern* has a slash  
8330 appended.8331 GLOB\_NOCHECK If *pattern* does not match any pathname, then return a list consisting of  
8332 only *pattern*.

8333 GLOB\_NOESCAPE Disable backslash escaping.

8334 GLOB\_NOSORT Do not sort the pathnames returned.

8335 The following constants shall be defined as error return values:

8336 GLOB\_ABORTED The scan was stopped because GLOB\_ERR was set or (\*errfunc())  
8337 returned non-zero.8338 GLOB\_NOMATCH The pattern does not match any existing pathname, and  
8339 GLOB\_NOCHECK was not set in *flags*.

8340 GLOB\_NOSPACE An attempt to allocate memory failed.

8341 OB XSI GLOB\_NOSYS Reserved.

8342 The following shall be declared as functions and may also be defined as macros. Function  
8343 prototypes shall be provided.8344 int glob(const char \*restrict, int, int (\*)(const char \*, int), 1  
8345 glob\_t \*restrict); 1

8346 void globfree(glob\_t \*);

8347 The implementation may define additional macros or constants using names beginning with  
8348 GLOB\_.

8349 **APPLICATION USAGE**

8350 None.

8351 **RATIONALE**

8352 None.

8353 **FUTURE DIRECTIONS**

8354 None.

8355 **SEE ALSO**

8356 The System Interfaces volume of IEEE Std 1003.1-2001, *glob()*, the Shell and Utilities volume of  
8357 IEEE Std 1003.1-2001

8358 **CHANGE HISTORY**

8359 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8360 **Issue 6**

8361 The **restrict** keyword is added to the prototype for *glob()*.

8362 The constant GLOB\_NOSYS is marked obsolescent.

8363 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the *glob()* 1  
8364 prototype definition by removing the **restrict** qualifier from the function pointer argument. 1

8365 **NAME**

8366       grp.h — group structure

8367 **SYNOPSIS**

8368       #include &lt;grp.h&gt;

8369 **DESCRIPTION**8370       The <grp.h> header shall declare the structure **group** which shall include the following  
8371       members:

8372       char    \*gr\_name   The name of the group.  
 8373       gid\_t   gr\_gid    Numerical group ID.  
 8374       char   \*\*gr\_mem    Pointer to a null-terminated array of character  
 8375       pointers to member names.

8376       The **gid\_t** type shall be defined as described in <sys/types.h>.8377       The following shall be declared as functions and may also be defined as macros. Function  
8378       prototypes shall be provided.

```

8379       struct group  *getgrgid(gid_t);
8380       struct group  *getgrnam(const char *);
8381 TSF       int        getgrgid_r(gid_t, struct group *, char *,
8382                               size_t, struct group **);
8383       int        getgrnam_r(const char *, struct group *, char *,
8384                               size_t , struct group **);
8385 XSI       struct group *getgrent(void);
8386       void       endgrent(void);
8387       void       setgrent(void);
8388
```

8389 **APPLICATION USAGE**

8390       None.

8391 **RATIONALE**

8392       None.

8393 **FUTURE DIRECTIONS**

8394       None.

8395 **SEE ALSO**8396       <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *endgrent()*, *getgrgid()*,  
8397       *getgrnam()*8398 **CHANGE HISTORY**

8399       First released in Issue 1.

8400 **Issue 5**

8401       The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8402 **Issue 6**8403       The following new requirements on POSIX implementations derive from alignment with the  
8404       Single UNIX Specification:

- 8405       • The definition of **gid\_t** is mandated.
- 8406       • The *getgrgid\_r()* and *getgrnam\_r()* functions are marked as part of the Thread-Safe Functions  
8407       option.

8408 **NAME**

8409 iconv.h — codeset conversion facility

8410 **SYNOPSIS**

8411 XSI `#include <iconv.h>`

8412

8413 **DESCRIPTION**

8414 The <iconv.h> header shall define the following type:

8415 **iconv\_t** Identifies the conversion from one codeset to another.

8416 The following shall be declared as functions and may also be defined as macros. Function  
8417 prototypes shall be provided.

```
8418 iconv_t iconv_open(const char *, const char *);
8419 size_t iconv(iconv_t, char **restrict, size_t *restrict,
8420             char **restrict, size_t *restrict);
8421 int iconv_close(iconv_t);
```

8422 **APPLICATION USAGE**

8423 None.

8424 **RATIONALE**

8425 None.

8426 **FUTURE DIRECTIONS**

8427 None.

8428 **SEE ALSO**

8429 The System Interfaces volume of IEEE Std 1003.1-2001, *iconv()*, *iconv\_close()*, *iconv\_open()*

8430 **CHANGE HISTORY**

8431 First released in Issue 4.

8432 **Issue 6**

8433 The **restrict** keyword is added to the prototype for *iconv()*.

8434 **NAME**

8435 inttypes.h — fixed size integer types

8436 **SYNOPSIS**

8437 #include &lt;inttypes.h&gt;

8438 **DESCRIPTION**

8439 **CX** Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

8443 The &lt;inttypes.h&gt; header shall include the &lt;stdint.h&gt; header.

8444 The &lt;inttypes.h&gt; header shall include a definition of at least the following type:

8445 **imaxdiv\_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

8446 The following macros shall be defined. Each expands to a character string literal containing a conversion specifier, possibly modified by a length modifier, suitable for use within the *format* argument of a formatted input/output function when converting the corresponding integer type. These macros have the general form of PRI (character string literals for the *fprintf()* and *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()* family of functions), followed by the conversion specifier, followed by a name corresponding to a similar type name in <stdint.h>. In these names, *N* represents the width of the type as described in <stdint.h>. For example, *PRIdFAST32* can be used in a format string to print the value of an integer of type *int\_fast32\_t*.

8455 The *fprintf()* macros for signed integers are:

8456	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
8457	PRiN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR

8458 The *fprintf()* macros for unsigned integers are:

8459	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8460	PRiUN	PRiULEASTN	PRiUFASTN	PRiUMAX	PRiUPTR
8461	PRIxN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR
8462	PRIXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR

8463 The *fscanf()* macros for signed integers are:

8464	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
8465	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

8466 The *fscanf()* macros for unsigned integers are:

8467	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
8468	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
8469	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

8470 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* and *fwprintf()* macros shall be defined and the corresponding *fscanf()* and *fwscanf()* macros shall be defined unless the implementation does not have a suitable modifier for the type.

8473 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
8475 intmax_t imaxabs(intmax_t);
8476 imaxdiv_t imaxdiv(intmax_t, intmax_t);
8477 intmax_t strtoumax(const char *restrict, char **restrict, int);
```



```
8478     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8479     intmax_t wcstoimax(const wchar_t *restrict, wchar_t **restrict, int);
8480     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
```

#### 8481 EXAMPLES

```
8482     #include <inttypes.h>
8483     #include <wchar.h>
8484     int main(void)
8485     {
8486         uintmax_t i = UINTMAX_MAX; // This type always exists.
8487         wprintf(L"The largest integer value is %020"
8488             PRIxMAX "\n", i);
8489         return 0;
8490     }
```

#### 8491 APPLICATION USAGE

8492 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent  
8493 across machines and independent of operating systems and other implementation  
8494 idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to  
8495 **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of  
8496 this header will greatly increase the portability of applications across platforms.

#### 8497 RATIONALE

8498 The ISO/IEC 9899:1990 standard specified that the language should support four signed and  
8499 unsigned integer data types—**char**, **short**, **int**, and **long**—but placed very little requirement on  
8500 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and  
8501 not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits  
8502 to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to  
8503 assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems  
8504 for users who migrate from one system to another which assigns different sizes to integer types,  
8505 because the ISO C standard integer promotion rule can produce silent changes unexpectedly.  
8506 The need for defining an extended integer type increased with the introduction of 64-bit  
8507 systems.

#### 8508 FUTURE DIRECTIONS

8509 Macro names beginning with PRI or SCN followed by any lowercase letter or 'x' may be added  
8510 to the macros defined in the <inttypes.h> header.

#### 8511 SEE ALSO

8512 The System Interfaces volume of IEEE Std 1003.1-2001, *imaxdiv()*

#### 8513 CHANGE HISTORY

8514 First released in Issue 5.

#### 8515 Issue 6

8516 The Open Group Base Resolution bwg97-006 is applied.

8517 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

8518 **NAME**

8519       iso646.h — alternative spellings

8520 **SYNOPSIS**

8521       #include &lt;iso646.h&gt;

8522 **DESCRIPTION**

8523 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 8524              conflict between the requirements described here and the ISO C standard is unintentional. This  
 8525              volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8526              The <iso646.h> header shall define the following eleven macros (on the left) that expand to the  
 8527              corresponding tokens (on the right):

8528              and               &amp;&amp;

8529              and\_eq           &amp;=

8530              bitand           &amp;

8531              bitor            |

8532              compl           ~

8533              not             !

8534              not\_eq           !=

8535              or               | |

8536              or\_eq           |=

8537              xor             ^

8538              xor\_eq           ^=

8539 **APPLICATION USAGE**

8540       None.

8541 **RATIONALE**

8542       None.

8543 **FUTURE DIRECTIONS**

8544       None.

8545 **SEE ALSO**

8546       None.

8547 **CHANGE HISTORY**

8548       First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8549 **NAME**

8550 langinfo.h — language information constants

8551 **SYNOPSIS**

8552 xSI #include <langinfo.h>

8553

8554 **DESCRIPTION**

8555 The <langinfo.h> header contains the constants used to identify items of *langinfo* data (see  
8556 *nl\_langinfo()*). The type of the constant, **nl\_item**, shall be defined as described in <nl\_types.h>.

8557 The following constants shall be defined. The entries under **Category** indicate in which  
8558 *setlocale()* category each item is defined.

8559

Constant	Category	Meaning
CODESET	LC_CTYPE	Codeset name.
D_T_FMT	LC_TIME	String for formatting date and time.
D_FMT	LC_TIME	Date format string.
T_FMT	LC_TIME	Time format string.
T_FMT_AMPM	LC_TIME	a.m. or p.m. time format string.
AM_STR	LC_TIME	Ante-meridiem affix.
PM_STR	LC_TIME	Post-meridiem affix.
DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
DAY_4	LC_TIME	Name of the fourth day of the week (for example, Wednesday).
DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
DAY_7	LC_TIME	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
MON_1	LC_TIME	Name of the first month of the year.
MON_2	LC_TIME	Name of the second month.
MON_3	LC_TIME	Name of the third month.
MON_4	LC_TIME	Name of the fourth month.
MON_5	LC_TIME	Name of the fifth month.
MON_6	LC_TIME	Name of the sixth month.
MON_7	LC_TIME	Name of the seventh month.
MON_8	LC_TIME	Name of the eighth month.
MON_9	LC_TIME	Name of the ninth month.
MON_10	LC_TIME	Name of the tenth month.
MON_11	LC_TIME	Name of the eleventh month.
MON_12	LC_TIME	Name of the twelfth month.

8596

8597

8598

8599

8600

8601

8602

8603

8604

8605

8606

8607

8608

8609

8610

8611

8612

8613

8614

8615

8616

8617

8618

8619

8620

8621

8622

8623

Constant	Category	Meaning
ABMON_1	LC_TIME	Abbreviated name of the first month.
ABMON_2	LC_TIME	Abbreviated name of the second month.
ABMON_3	LC_TIME	Abbreviated name of the third month.
ABMON_4	LC_TIME	Abbreviated name of the fourth month.
ABMON_5	LC_TIME	Abbreviated name of the fifth month.
ABMON_6	LC_TIME	Abbreviated name of the sixth month.
ABMON_7	LC_TIME	Abbreviated name of the seventh month.
ABMON_8	LC_TIME	Abbreviated name of the eighth month.
ABMON_9	LC_TIME	Abbreviated name of the ninth month.
ABMON_10	LC_TIME	Abbreviated name of the tenth month.
ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
ERA	LC_TIME	Era description segments.
ERA_D_FMT	LC_TIME	Era date format string.
ERA_D_T_FMT	LC_TIME	Era date and time format string.
ERA_T_FMT	LC_TIME	Era time format string.
ALT_DIGITS	LC_TIME	Alternative symbols for digits.
RADIXCHAR	LC_NUMERIC	Radix character.
THOUSEP	LC_NUMERIC	Separator for thousands.
YESEXPR	LC_MESSAGES	Affirmative response expression.
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Local currency symbol, preceded by '−' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string ("").

1

1

1

8624

8625

If the locale's values for **p\_cs\_precedes** and **n\_cs\_precedes** do not match, the value of *nl\_langinfo*(CRNCYSTR) is unspecified.

8626

8627

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

8628

```
char *nl_langinfo(nl_item);
```

8629

Inclusion of the <langinfo.h> header may also make visible all symbols from <nl\_types.h>.

## 8630 APPLICATION USAGE

8631

8632

8633

8634

8635

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of *langinfo* data. In particular, the *strftime*() function should be used to access date and time information defined in category *LC\_TIME*. The *localeconv*() function should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and *CRNCYSTR*.

## 8636 RATIONALE

8637

None.

## 8638 FUTURE DIRECTIONS

8639

None.

## 8640 SEE ALSO

8641

8642

The System Interfaces volume of IEEE Std 1003.1-2001, *nl\_langinfo*(), *localeconv*(), *strfmon*(), *strftime*(), Chapter 7 (on page 123)

8643 **CHANGE HISTORY**

8644 First released in Issue 2.

8645 **Issue 5**

8646 The constants YESSTR and NOSTR are marked LEGACY.

8647 **Issue 6**

8648 The constants YESSTR and NOSTR are removed.

8649 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to 1  
 8650 the“Meaning” column entry for the CRNCYSTR constant. This change is to accommodate 1  
 8651 historic practice. 1

8652 **NAME**

8653 libgen.h — definitions for pattern matching functions

8654 **SYNOPSIS**8655 XSI `#include <libgen.h>`

8656

8657 **DESCRIPTION**8658 The following shall be declared as functions and may also be defined as macros. Function  
8659 prototypes shall be provided.8660 `char *basename(char *);`8661 `char *dirname(char *);`8662 **APPLICATION USAGE**

8663 None.

8664 **RATIONALE**

8665 None.

8666 **FUTURE DIRECTIONS**

8667 None.

8668 **SEE ALSO**8669 The System Interfaces volume of IEEE Std 1003.1-2001, *basename()*, *dirname()*8670 **CHANGE HISTORY**

8671 First released in Issue 4, Version 2.

8672 **Issue 5**8673 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first  
8674 argument is of type **char** \* rather than **const char** \*.8675 **Issue 6**8676 The `__loc1` symbol and the *regcmp()* and *regex()* functions are removed.

8677 **NAME**

8678 limits.h — implementation-defined constants

8679 **SYNOPSIS**

8680 #include <limits.h>

8681 **DESCRIPTION**

8682 CX Some of the functionality described on this reference page extends the ISO C standard.  
8683 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
8684 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
8685 symbols in this header.

8686 CX Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such  
8687 symbols are not shown as CX shaded.

8688 The <limits.h> header shall define various symbolic names. Different categories of names are  
8689 described below.

8690 The names represent various limits on resources that the implementation imposes on  
8691 applications.

8692 Implementations may choose any appropriate value for each limit, provided it is not more  
8693 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names  
8694 beginning with \_POSIX may be found in <unistd.h>.

8695 Applications should not assume any particular value for a limit. To achieve maximum  
8696 portability, an application should not require more resource than the Minimum Acceptable  
8697 Value quantity. However, an application wishing to avail itself of the full amount of a resource  
8698 available on an implementation may make use of the value given in <limits.h> on that  
8699 particular implementation, by using the symbolic names listed below. It should be noted,  
8700 however, that many of the listed limits are not invariant, and at runtime, the value of the limit  
8701 may differ from those given in this header, for the following reasons:

- 8702 • The limit is pathname-dependent.
- 8703 • The limit differs between the compile and runtime machines.

8704 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to  
8705 determine the actual value of a limit at runtime.

8706 The items in the list ending in \_MIN give the most negative values that the mathematical types  
8707 are guaranteed to be capable of representing. Numbers of a more negative value may be  
8708 supported on some implementations, as indicated by the <limits.h> header on the  
8709 implementation, but applications requiring such numbers are not guaranteed to be portable to  
8710 all implementations. For positive constants ending in \_MIN, this indicates the minimum  
8711 acceptable value.

8712 **Runtime Invariant Values (Possibly Indeterminate)**

8713 A definition of one of the symbolic names in the following list shall be omitted from <limits.h>  
8714 on specific implementations where the corresponding value is equal to or greater than the stated  
8715 minimum, but is unspecified.

8716 This indetermination might depend on the amount of available memory space on a specific  
8717 instance of a specific implementation. The actual value supported by a specific instance shall be  
8718 provided by the *sysconf()* function.

8719 AIO {AIO\_LISTIO\_MAX}  
8720 Maximum number of I/O operations in a single list I/O call supported by the

8721		implementation.
8722		Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}
8723	AIO	{AIO_MAX}
8724		Maximum number of outstanding asynchronous I/O operations supported by the
8725		implementation.
8726		Minimum Acceptable Value: {_POSIX_AIO_MAX}
8727	AIO	{AIO_PRIO_DELTA_MAX}
8728		The maximum amount by which a process can decrease its asynchronous I/O priority level
8729		from its own scheduling priority.
8730		Minimum Acceptable Value: 0
8731		{ARG_MAX}
8732		Maximum length of argument to the <i>exec</i> functions including environment data.
8733		Minimum Acceptable Value: {_POSIX_ARG_MAX}
8734	XSI	{ATEXIT_MAX}
8735		Maximum number of functions that may be registered with <i>atexit()</i> .
8736		Minimum Acceptable Value: 32
8737		{CHILD_MAX}
8738		Maximum number of simultaneous processes per real user ID.
8739		Minimum Acceptable Value: {_POSIX_CHILD_MAX}
8740	TMR	{DELAYTIMER_MAX}
8741		Maximum number of timer expiration overruns.
8742		Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}
8743		{HOST_NAME_MAX}
8744		Maximum length of a host name (not including the terminating null) as returned from the
8745		<i>gethostname()</i> function.
8746		Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}
8747	XSI	{IOV_MAX}
8748		Maximum number of <i>iovec</i> structures that one process has available for use with <i>readv()</i> or
8749		<i>writev()</i> .
8750		Minimum Acceptable Value: {_XOPEN_IOV_MAX}
8751		{LOGIN_NAME_MAX}
8752		Maximum length of a login name.
8753		Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}
8754	MSG	{MQ_OPEN_MAX}
8755		The maximum number of open message queue descriptors a process may hold.
8756		Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}
8757	MSG	{MQ_PRIO_MAX}
8758		The maximum number of message priorities supported by the implementation.
8759		Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}
8760		{OPEN_MAX}
8761		Maximum number of files that one process can have open at any one time.
8762		Minimum Acceptable Value: {_POSIX_OPEN_MAX}
8763		{PAGESIZE}
8764		Size in bytes of a page.
8765		Minimum Acceptable Value: 1



8766	XSI	{PAGE_SIZE}
8767		Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
8768		defined with the same value.
8769	THR	{PTHREAD_DESTRUCTOR_ITERATIONS}
8770		Maximum number of attempts made to destroy a thread's thread-specific data values on
8771		thread exit.
8772		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8773	THR	{PTHREAD_KEYS_MAX}
8774		Maximum number of data keys that can be created by a process.
8775		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
8776	THR	{PTHREAD_STACK_MIN}
8777		Minimum size in bytes of thread stack storage.
8778		Minimum Acceptable Value: 0
8779	THR	{PTHREAD_THREADS_MAX}
8780		Maximum number of threads that can be created per process.
8781		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
8782		{RE_DUP_MAX}
8783		The number of repeated occurrences of a BRE permitted by the <i>regex</i> ( <i>c</i> ) and <i>regcomp</i> ( <i>c</i> )
8784		functions when using the interval notation <i>{\{m,n\}}</i> ; see Section 9.3.6 (on page 174).
8785		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8786	RTS	{RTSIG_MAX}
8787		Maximum number of realtime signals reserved for application use in this implementation.
8788		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
8789	SEM	{SEM_NSEMS_MAX}
8790		Maximum number of semaphores that a process may have.
8791		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
8792	SEM	{SEM_VALUE_MAX}
8793		The maximum value a semaphore may have.
8794		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
8795	RTS	{SIGQUEUE_MAX}
8796		Maximum number of queued signals that a process may send and have pending at the
8797		receiver(s) at any time.
8798		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
8799	SS TSP	{SS_REPL_MAX}
8800		The maximum number of replenishment operations that may be simultaneously pending
8801		for a particular sporadic server scheduler.
8802		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}
8803		{STREAM_MAX}
8804		The number of streams that one process can have open at one time. If defined, it has the
8805		same value as {FOPEN_MAX} (see <stdio.h>).
8806		Minimum Acceptable Value: {_POSIX_STREAM_MAX}
8807		{SYMLOOP_MAX}
8808		Maximum number of symbolic links that can be reliably traversed in the resolution of a
8809		pathname in the absence of a loop.
8810		Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

8811	TMR	{TIMER_MAX}
8812		Maximum number of timers per process supported by the implementation.
8813		Minimum Acceptable Value: {_POSIX_TIMER_MAX}
8814	TRC	{TRACE_EVENT_NAME_MAX}
8815		Maximum length of the trace event name.
8816		Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}
8817	TRC	{TRACE_NAME_MAX}
8818		Maximum length of the trace generation version string or of the trace stream name.
8819		Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}
8820	TRC	{TRACE_SYS_MAX}
8821		Maximum number of trace streams that may simultaneously exist in the system.
8822		Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}
8823	TRC	{TRACE_USER_EVENT_MAX}
8824		Maximum number of user trace event type identifiers that may simultaneously exist in a
8825		traced process, including the predefined user trace event
8826		POSIX_TRACE_UNNAMED_USER_EVENT.
8827		Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}
8828		{TTY_NAME_MAX}
8829		Maximum length of terminal device name.
8830		Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}
8831		{TZNAME_MAX}
8832		Maximum number of bytes supported for the name of a timezone (not of the <i>TZ</i> variable).
8833		Minimum Acceptable Value: {_POSIX_TZNAME_MAX}
8834	<b>Note:</b>	The length given by {TZNAME_MAX} does not include the quoting characters mentioned in
8835		Section 8.3 (on page 165).

## 8836 Pathname Variable Values

8837 The values in the following list may be constants within an implementation or may vary from  
 8838 one pathname to another. For example, file systems or directories may have different  
 8839 characteristics.

8840 A definition of one of the values shall be omitted from the <limits.h> header on specific  
 8841 implementations where the corresponding value is equal to or greater than the stated minimum,  
 8842 but where the value can vary depending on the file to which it is applied. The actual value  
 8843 supported for a specific pathname shall be provided by the *pathconf()* function.

8844 {FILESIZEBITS}

8845 Minimum number of bits needed to represent, as a signed integer value, the maximum size  
 8846 of a regular file allowed in the specified directory.

8847 Minimum Acceptable Value: 32

8848 {LINK\_MAX}

8849 Maximum number of links to a single file.

8850 Minimum Acceptable Value: {\_POSIX\_LINK\_MAX}

8851 {MAX\_CANON}

8852 Maximum number of bytes in a terminal canonical input line.

8853 Minimum Acceptable Value: {\_POSIX\_MAX\_CANON}

8854 {MAX\_INPUT}

8855 Minimum number of bytes for which space is available in a terminal input queue; therefore,

8856 the maximum number of bytes a conforming application may require to be typed as input  
 8857 before reading them.  
 8858 Minimum Acceptable Value: {\_POSIX\_MAX\_INPUT}

8859 {NAME\_MAX}  
 8860 Maximum number of bytes in a filename (not including terminating null).  
 8861 Minimum Acceptable Value: {\_POSIX\_NAME\_MAX}  
 8862 XSI Minimum Acceptable Value: {\_XOPEN\_NAME\_MAX}

8863 {PATH\_MAX}  
 8864 Maximum number of bytes in a pathname, including the terminating null character.  
 8865 Minimum Acceptable Value: {\_POSIX\_PATH\_MAX}  
 8866 XSI Minimum Acceptable Value: {\_XOPEN\_PATH\_MAX}

8867 {PIPE\_BUF}  
 8868 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.  
 8869 Minimum Acceptable Value: {\_POSIX\_PIPE\_BUF}

8870 ADV {POSIX\_ALLOC\_SIZE\_MIN}  
 8871 Minimum number of bytes of storage actually allocated for any portion of a file.  
 8872 Minimum Acceptable Value: Not specified.

8873 ADV {POSIX\_REC\_INCR\_XFER\_SIZE}  
 8874 Recommended increment for file transfer sizes between the  
 8875 {POSIX\_REC\_MIN\_XFER\_SIZE} and {POSIX\_REC\_MAX\_XFER\_SIZE} values.  
 8876 Minimum Acceptable Value: Not specified.

8877 ADV {POSIX\_REC\_MAX\_XFER\_SIZE}  
 8878 Maximum recommended file transfer size.  
 8879 Minimum Acceptable Value: Not specified.

8880 ADV {POSIX\_REC\_MIN\_XFER\_SIZE}  
 8881 Minimum recommended file transfer size.  
 8882 Minimum Acceptable Value: Not specified.

8883 ADV {POSIX\_REC\_XFER\_ALIGN}  
 8884 Recommended file transfer buffer alignment.  
 8885 Minimum Acceptable Value: Not specified.

8886 {SYMLINK\_MAX}  
 8887 Maximum number of bytes in a symbolic link.  
 8888 Minimum Acceptable Value: {\_POSIX\_SYMLINK\_MAX}

### Runtime Increaseable Values

8890 The magnitude limitations in the following list shall be fixed by specific implementations. An  
 8891 application should assume that the value supplied by <limits.h> in a specific implementation is  
 8892 the minimum that pertains whenever the application is run under that implementation. A  
 8893 specific instance of a specific implementation may increase the value relative to that supplied by  
 8894 <limits.h> for that implementation. The actual value supported by a specific instance shall be  
 8895 provided by the *sysconf()* function.

8896 {BC\_BASE\_MAX}  
 8897 Maximum *obase* values allowed by the *bc* utility.  
 8898 Minimum Acceptable Value: {\_POSIX2\_BC\_BASE\_MAX}

8899 {BC\_DIM\_MAX}  
 8900 Maximum number of elements permitted in an array by the *bc* utility.

8901		Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}
8902		{BC_SCALE_MAX}
8903		Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
8904		Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}
8905		{BC_STRING_MAX}
8906		Maximum length of a string constant accepted by the <i>bc</i> utility.
8907		Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}
8908		{CHARCLASS_NAME_MAX}
8909		Maximum number of bytes in a character class name.
8910		Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}
8911		{COLL_WEIGHTS_MAX}
8912		Maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b>
8913		keyword in the locale definition file; see Chapter 7 (on page 123).
8914		Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}
8915		{EXPR_NEST_MAX}
8916		Maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.
8917		Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}
8918		{LINE_MAX}
8919		Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
8920		standard input or another file), when the utility is described as processing text files. The
8921		length includes room for the trailing <newline>.
8922		Minimum Acceptable Value: {_POSIX2_LINE_MAX}
8923		{NGROUPS_MAX}
8924		Maximum number of simultaneous supplementary group IDs per process.
8925		Minimum Acceptable Value: {_POSIX2_NGROUPS_MAX}
8926		{RE_DUP_MAX}
8927		Maximum number of repeated occurrences of a regular expression permitted when using
8928		the interval notation $\{m,n\}$ ; see Chapter 9 (on page 169).
8929		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8930		<b>Maximum Values</b>
8931	TMR	The symbolic constants in the following list shall be defined in <limits.h> with the values
8932		shown. These are symbolic names for the most restrictive value for certain features on an
8933		implementation supporting the Timers option. A conforming implementation shall provide
8934		values no larger than these values. A conforming application must not require a smaller value
8935		for correct operation.
8936	TMR	{_POSIX_CLOCKRES_MIN}
8937		The resolution of the CLOCK_REALTIME clock, in nanoseconds.
8938		Value: 20 000 000
8939	MON	If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC
8940		clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

8941 **Minimum Values**

8942 The symbolic constants in the following list shall be defined in <limits.h> with the values  
 8943 shown. These are symbolic names for the most restrictive value for certain features on an  
 8944 implementation conforming to this volume of IEEE Std 1003.1-2001. Related symbolic constants  
 8945 are defined elsewhere in this volume of IEEE Std 1003.1-2001 which reflect the actual  
 8946 implementation and which need not be as restrictive. A conforming implementation shall  
 8947 provide values at least this large. A strictly conforming application must not require a larger  
 8948 value for correct operation.

8949 AIO { \_POSIX\_AIO\_LISTIO\_MAX}  
 8950 The number of I/O operations that can be specified in a list I/O call.  
 8951 Value: 2

8952 AIO { \_POSIX\_AIO\_MAX}  
 8953 The number of outstanding asynchronous I/O operations.  
 8954 Value: 1

8955 { \_POSIX\_ARG\_MAX}  
 8956 Maximum length of argument to the *exec* functions including environment data.  
 8957 Value: 4 096

8958 { \_POSIX\_CHILD\_MAX}  
 8959 Maximum number of simultaneous processes per real user ID.  
 8960 Value: 25

1

8961 TMR { \_POSIX\_DELAYTIMER\_MAX}  
 8962 The number of timer expiration overruns.  
 8963 Value: 32

8964 { \_POSIX\_HOST\_NAME\_MAX}  
 8965 Maximum length of a host name (not including the terminating null) as returned from the  
 8966 *gethostname()* function.  
 8967 Value: 255

8968 { \_POSIX\_LINK\_MAX}  
 8969 Maximum number of links to a single file.  
 8970 Value: 8

8971 { \_POSIX\_LOGIN\_NAME\_MAX}  
 8972 The size of the storage required for a login name, in bytes, including the terminating null.  
 8973 Value: 9

8974 { \_POSIX\_MAX\_CANON}  
 8975 Maximum number of bytes in a terminal canonical input queue.  
 8976 Value: 255

8977 { \_POSIX\_MAX\_INPUT}  
 8978 Maximum number of bytes allowed in a terminal input queue.  
 8979 Value: 255

8980 MSG { \_POSIX\_MQ\_OPEN\_MAX}  
 8981 The number of message queues that can be open for a single process.  
 8982 Value: 8

8983 MSG { \_POSIX\_MQ\_PRIO\_MAX}  
 8984 The maximum number of message priorities supported by the implementation.  
 8985 Value: 32

8986		{_POSIX_NAME_MAX}
8987		Maximum number of bytes in a filename (not including terminating null).
8988		Value: 14
8989		{_POSIX_NGROUPS_MAX}
8990		Maximum number of simultaneous supplementary group IDs per process.
8991		Value: 8
8992		{_POSIX_OPEN_MAX}
8993		Maximum number of files that one process can have open at any one time.
8994		Value: 20
8995		{_POSIX_PATH_MAX}
8996		Maximum number of bytes in a pathname.
8997		Value: 256
8998		{_POSIX_PIPE_BUF}
8999		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
9000		Value: 512
9001		{_POSIX_RE_DUP_MAX}
9002		The number of repeated occurrences of a BRE permitted by the <i>regex</i> ( <i>re</i> ) and <i>regcomp</i> ( <i>re</i> )
9003		functions when using the interval notation <i>{(m,n)}</i> ; see Section 9.3.6 (on page 174).
9004		Value: 255
9005	RTS	{_POSIX_RTSIG_MAX}
9006		The number of realtime signal numbers reserved for application use.
9007		Value: 8
9008	SEM	{_POSIX_SEM_NSEMS_MAX}
9009		The number of semaphores that a process may have.
9010		Value: 256
9011	SEM	{_POSIX_SEM_VALUE_MAX}
9012		The maximum value a semaphore may have.
9013		Value: 32 767
9014	RTS	{_POSIX_SIGQUEUE_MAX}
9015		The number of queued signals that a process may send and have pending at the receiver(s)
9016		at any time.
9017		Value: 32
9018		{_POSIX_SSIZE_MAX}
9019		The value that can be stored in an object of type <i>ssize_t</i> .
9020		Value: 32 767
9021		{_POSIX_STREAM_MAX}
9022		The number of streams that one process can have open at one time.
9023		Value: 8
9024	SS TSP	{_POSIX_SS_REPL_MAX}
9025		The number of replenishment operations that may be simultaneously pending for a
9026		particular sporadic server scheduler.
9027		Value: 4
9028		{_POSIX_SYMLINK_MAX}
9029		The number of bytes in a symbolic link.
9030		Value: 255

9031		{_POSIX_SYMLINK_MAX}
9032		The number of symbolic links that can be traversed in the resolution of a pathname in the
9033		absence of a loop.
9034		Value: 8
9035	THR	{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
9036		The number of attempts made to destroy a thread's thread-specific data values on thread
9037		exit.
9038		Value: 4
9039	THR	{_POSIX_THREAD_KEYS_MAX}
9040		The number of data keys per process.
9041		Value: 128
9042	THR	{_POSIX_THREAD_THREADS_MAX}
9043		The number of threads per process.
9044		Value: 64
9045	TMR	{_POSIX_TIMER_MAX}
9046		The per-process number of timers.
9047		Value: 32
9048	TRC	{_POSIX_TRACE_EVENT_NAME_MAX}
9049		The length in bytes of a trace event name.
9050		Value: 30
9051	TRC	{_POSIX_TRACE_NAME_MAX}
9052		The length in bytes of a trace generation version string or a trace stream name.
9053		Value: 8
9054	TRC	{_POSIX_TRACE_SYS_MAX}
9055		The number of trace streams that may simultaneously exist in the system.
9056		Value: 8
9057	TRC	{_POSIX_TRACE_USER_EVENT_MAX}
9058		The number of user trace event type identifiers that may simultaneously exist in a traced
9059		process, including the predefined user trace event
9060		POSIX_TRACE_UNNAMED_USER_EVENT.
9061		Value: 32
9062		{_POSIX_TTY_NAME_MAX}
9063		The size of the storage required for a terminal device name, in bytes, including the
9064		terminating null.
9065		Value: 9
9066		{_POSIX_TZNAME_MAX}
9067		Maximum number of bytes supported for the name of a timezone (not of the <i>TZ</i> variable).
9068		Value: 6
9069		<b>Note:</b> The length given by {_POSIX_TZNAME_MAX} does not include the quoting characters
9070		mentioned in Section 8.3 (on page 165).
9071		{_POSIX2_BC_BASE_MAX}
9072		Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
9073		Value: 99
9074		{_POSIX2_BC_DIM_MAX}
9075		Maximum number of elements permitted in an array by the <i>bc</i> utility.
9076		Value: 2 048

9077		{_POSIX2_BC_SCALE_MAX}
9078		Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
9079		Value: 99
9080		{_POSIX2_BC_STRING_MAX}
9081		Maximum length of a string constant accepted by the <i>bc</i> utility.
9082		Value: 1 000
9083		{_POSIX2_CHARCLASS_NAME_MAX}
9084		Maximum number of bytes in a character class name.
9085		Value: 14
9086		{_POSIX2_COLL_WEIGHTS_MAX}
9087		Maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b>
9088		keyword in the locale definition file; see Chapter 7 (on page 123).
9089		Value: 2
9090		{_POSIX2_EXPR_NEST_MAX}
9091		Maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.
9092		Value: 32
9093		{_POSIX2_LINE_MAX}
9094		Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
9095		standard input or another file), when the utility is described as processing text files. The
9096		length includes room for the trailing <newline>.
9097		Value: 2 048
9098		{_POSIX2_RE_DUP_MAX}
9099		Maximum number of repeated occurrences of a regular expression permitted when using
9100		the interval notation $\{m,n\}$ ; see Chapter 9 (on page 169).
9101		Value: 255
9102	XSI	{_XOPEN_IOV_MAX}
9103		Maximum number of <b>iovec</b> structures that one process has available for use with <i>readv()</i> or
9104		<i>writev()</i> .
9105		Value: 16
9106	XSI	{_XOPEN_NAME_MAX}
9107		Maximum number of bytes in a filename (not including the terminating null).
9108		Value: 255
9109	XSI	{_XOPEN_PATH_MAX}
9110		Maximum number of bytes in a pathname.
9111		Value: 1 024
9112		<b>Numerical Limits</b>
9113		The values in the following lists shall be defined in <limits.h> and are constant expressions
9114	XSI	suitable for use in <b>#if</b> preprocessing directives. Moreover, except for {CHAR_BIT}, {DBL_DIG},
9115		{DBL_MAX}, {FLT_DIG}, {FLT_MAX}, {LONG_BIT}, {WORD_BIT}, and {MB_LEN_MAX}, the
9116		symbolic names are defined as expressions of the correct type.
9117		If the value of an object of type <b>char</b> is treated as a signed integer when used in an expression,
9118		the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
9119		is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
9120		of {CHAR_MAX} is the same as that of {UCHAR_MAX}.



9121		{CHAR_BIT}	
9122		Number of bits in a type <b>char</b> .	
9123	CX	Value: 8	
9124		{CHAR_MAX}	
9125		Maximum value of type <b>char</b> .	
9126		Value: {UCHAR_MAX} or {SCHAR_MAX}	
9127		{CHAR_MIN}	
9128		Minimum value of type <b>char</b> .	
9129		Value: {SCHAR_MIN} or 0	
9130		{INT_MAX}	
9131		Maximum value of an <b>int</b> .	
9132	CX	Minimum Acceptable Value: 2 147 483 647	2
9133	XSI	{LONG_BIT}	
9134		Number of bits in a <b>long</b> .	
9135		Minimum Acceptable Value: 32	
9136		{LONG_MAX}	
9137		Maximum value of a <b>long</b> .	
9138		Minimum Acceptable Value: +2 147 483 647	
9139		{MB_LEN_MAX}	
9140		Maximum number of bytes in a character, for any supported locale.	
9141		Minimum Acceptable Value: 1	
9142		{SCHAR_MAX}	
9143		Maximum value of type <b>signed char</b> .	
9144	CX	Value: +127	
9145		{SHRT_MAX}	
9146		Maximum value of type <b>short</b> .	
9147		Minimum Acceptable Value: +32 767	
9148		{SSIZE_MAX}	
9149		Maximum value of an object of type <b>ssize_t</b> .	
9150		Minimum Acceptable Value: {_POSIX_SSIZE_MAX}	
9151		{UCHAR_MAX}	
9152		Maximum value of type <b>unsigned char</b> .	
9153	CX	Value: 255	
9154		{UINT_MAX}	
9155		Maximum value of type <b>unsigned</b> .	
9156	CX	Minimum Acceptable Value: 4 294 967 295	2
9157		{ULONG_MAX}	
9158		Maximum value of type <b>unsigned long</b> .	
9159		Minimum Acceptable Value: 4 294 967 295	
9160		{USHRT_MAX}	
9161		Maximum value for a type <b>unsigned short</b> .	
9162		Minimum Acceptable Value: 65 535	
9163	XSI	{WORD_BIT}	
9164		Number of bits in a type <b>int</b> .	
9165		Minimum Acceptable Value: 32	2

9166		{INT_MIN}	
9167		Minimum value of type <b>int</b> .	
9168	CX	Maximum Acceptable Value: -2 147 483 647	2
9169		{LONG_MIN}	
9170		Minimum value of type <b>long</b> .	
9171		Maximum Acceptable Value: -2 147 483 647	
9172		{SCHAR_MIN}	
9173		Minimum value of type <b>signed char</b> .	
9174	CX	Value: -128	
9175		{SHRT_MIN}	
9176		Minimum value of type <b>short</b> .	
9177		Maximum Acceptable Value: -32 767	
9178		{LLONG_MIN}	
9179		Minimum value of type <b>long long</b> .	
9180		Maximum Acceptable Value: -9 223 372 036 854 775 807	
9181		{LLONG_MAX}	
9182		Maximum value of type <b>long long</b> .	
9183		Minimum Acceptable Value: +9 223 372 036 854 775 807	
9184		{ULLONG_MAX}	
9185		Maximum value of type <b>unsigned long long</b> .	
9186		Minimum Acceptable Value: 18 446 744 073 709 551 615	
9187		<b>Other Invariant Values</b>	
9188	XSI	The following constants shall be defined on all implementations in <limits.h>:	
9189	XSI	{NL_ARGMAX}	2
9190		Maximum value of <i>digit</i> in calls to the <i>printf()</i> and <i>scanf()</i> functions.	
9191		Minimum Acceptable Value: 9	
9192	XSI	{NL_LANGMAX}	
9193		Maximum number of bytes in a <i>LANG</i> name.	
9194		Minimum Acceptable Value: 14	
9195	XSI	{NL_MSGMAX}	
9196		Maximum message number.	
9197		Minimum Acceptable Value: 32 767	
9198	XSI	{NL_NMAX}	
9199		Maximum number of bytes in an N-to-1 collation mapping.	
9200		Minimum Acceptable Value: No guaranteed value across all conforming implementations.	
9201	XSI	{NL_SETMAX}	
9202		Maximum set number.	
9203		Minimum Acceptable Value: 255	
9204	XSI	{NL_TEXTMAX}	
9205		Maximum number of bytes in a message string.	
9206		Minimum Acceptable Value: {_POSIX2_LINE_MAX}	
9207	XSI	{NZERO}	
9208		Default process priority.	
9209		Minimum Acceptable Value: 20	

9210 **APPLICATION USAGE**

9211 None.

9212 **RATIONALE**

9213 A request was made to reduce the value of `{_POSIX_LINK_MAX}` from the value of 8 specified  
9214 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request  
9215 for several reasons:

- 9216 • They wanted to avoid making any changes to the standard that could break conforming  
9217 applications, and the requested change could have that effect.
- 9218 • The use of multiple hard links to a file cannot always be replaced with use of symbolic links.  
9219 Symbolic links are semantically different from hard links in that they associate a pathname  
9220 with another pathname rather than a pathname with a file. This has implications for access  
9221 control, file permanence, and transparency.
- 9222 • The original standard developers had considered the issue of allowing for implementations  
9223 that did not in general support hard links, and decided that this would reduce consensus on  
9224 the standard.

9225 Systems that support historical versions of the development option of the ISO POSIX-2 standard  
9226 retain the name `{_POSIX2_RE_DUP_MAX}` as an alias for `{_POSIX_RE_DUP_MAX}`.

9227 `{PATH_MAX}`

9228 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the  
9229 definition of pathname and the description of `{PATH_MAX}`, allowing application writers to  
9230 allocate either `{PATH_MAX}` or `{PATH_MAX}+1` bytes. The inconsistency has been  
9231 removed by correction to the `{PATH_MAX}` definition to include the null character. With  
9232 this change, applications that previously allocated `{PATH_MAX}` bytes will continue to  
9233 succeed.

9234 `{SYMLINK_MAX}`

9235 This symbol refers to space for data that is stored in the file system, as opposed to  
9236 `{PATH_MAX}` which is the length of a name that can be passed to a function. In some  
9237 existing implementations, the filenames pointed to by symbolic links are stored in the  
9238 inodes of the links, so it is important that `{SYMLINK_MAX}` not be constrained to be as  
9239 large as `{PATH_MAX}`.

9240 **FUTURE DIRECTIONS**

9241 None.

9242 **SEE ALSO**

9243 The System Interfaces volume of IEEE Std 1003.1-2001, *fpathconf()*, *pathconf()*, *sysconf()*

9244 **CHANGE HISTORY**

9245 First released in Issue 1.

9246 **Issue 5**

9247 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
9248 Threads Extension.

9249 `{FILESIZEBITS}` is added for the Large File Summit extensions.

9250 The minimum acceptable values for `{INT_MAX}`, `{INT_MIN}`, and `{UINT_MAX}` are changed to  
9251 make 32-bit values the minimum requirement.

9252 The entry is restructured to improve readability.

**Issue 6**

The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR\_MIN}, {INT\_MIN}, {LONG\_MIN}, {SCHAR\_MIN}, and {SHRT\_MIN} that these are maximum acceptable values.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The minimum value for {CHILD\_MAX} is 25. This is a FIPS requirement.
- The minimum value for {OPEN\_MAX} is 20. This is a FIPS requirement.
- The minimum value for {NGROUPS\_MAX} is 8. This is also a FIPS requirement.

Symbolic constants are added for {\_POSIX\_SYMLINK\_MAX}, {\_POSIX\_SYMLINK\_MAX}, {\_POSIX\_RE\_DUP\_MAX}, {RE\_DUP\_MAX}, {SYMLOOP\_MAX}, and {SYMLINK\_MAX}.

The following values are added for alignment with IEEE Std 1003.1d-1999:

```
{_POSIX_SS_REPL_MAX}
{SS_REPL_MAX}
{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}
```

Reference to CLOCK\_MONOTONIC is added in the description of {\_POSIX\_CLOCKRES\_MIN} for alignment with IEEE Std 1003.1j-2000.

The constants {LLONG\_MIN}, {LLONG\_MAX}, and {ULLONG\_MAX} are added for alignment with the ISO/IEC 9899:1999 standard.

The following values are added for alignment with IEEE Std 1003.1q-2000:

```
{_POSIX_TRACE_EVENT_NAME_MAX}
{_POSIX_TRACE_NAME_MAX}
{_POSIX_TRACE_SYS_MAX}
{_POSIX_TRACE_USER_EVENT_MAX}
{TRACE_EVENT_NAME_MAX}
{TRACE_NAME_MAX}
{TRACE_SYS_MAX}
{TRACE_USER_EVENT_MAX}
```

The new limits {\_XOPEN\_NAME\_MAX} and {\_XOPEN\_PATH\_MAX} are added as minimum values for {PATH\_MAX} and {NAME\_MAX} limits on XSI-conformant systems.

The legacy symbols {PASS\_MAX} and {TMP\_MAX} are removed.

The values for the limits {CHAR\_BIT}, {SCHAR\_MAX}, and {UCHAR\_MAX} are now required to be 8, +127, and 255, respectively.

The value for the limit {CHAR\_MAX} is now {UCHAR\_MAX} or {SCHAR\_MAX}.

The value for the limit {CHAR\_MIN} is now {SCHAR\_MIN} or zero.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of {\_POSIX\_CHILD\_MAX} from 6 to 25. This is for FIPS 151-2 alignment. 1 1

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/19 is applied, updating the values for {INT\_MAX}, {UINT\_MAX}, and {INT\_MIN} to be CX extensions over the ISO C standard, and 2 2

9296	correcting {WORD_BIT} from 16 to 32.	2
9297	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/20 is applied, removing	2
9298	{CHARCLASS_NAME_MAX} from the “Other Invariant Values” section (it also occurs under	2
9299	“Runtime Inceasable Values”).	2

9300 **NAME**

9301       locale.h — category macros

9302 **SYNOPSIS**

9303       #include &lt;locale.h&gt;

9304 **DESCRIPTION**

9305 CX       Some of the functionality described on this reference page extends the ISO C standard. Any  
 9306       conflict between the requirements described here and the ISO C standard is unintentional. This  
 9307       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

9308       The <locale.h> header shall provide a definition for **lconv** structure, which shall include at least  
 9309       the following members. (See the definitions of *LC\_MONETARY* in Section 7.3.3 (on page 142)  
 9310       and Section 7.3.4 (on page 145).)

```

9311         char    *currency_symbol
9312         char    *decimal_point
9313         char    frac_digits
9314         char    *grouping
9315         char    *int_curr_symbol
9316         char    int_frac_digits
9317         char    int_n_cs_precedes
9318         char    int_n_sep_by_space
9319         char    int_n_sign_posn
9320         char    int_p_cs_precedes
9321         char    int_p_sep_by_space
9322         char    int_p_sign_posn
9323         char    *mon_decimal_point
9324         char    *mon_grouping
9325         char    *mon_thousands_sep
9326         char    *negative_sign
9327         char    n_cs_precedes
9328         char    n_sep_by_space
9329         char    n_sign_posn
9330         char    *positive_sign
9331         char    p_cs_precedes
9332         char    p_sep_by_space
9333         char    p_sign_posn
9334         char    *thousands_sep
  
```

9335       The <locale.h> header shall define NULL (as defined in <stddef.h>) and at least the following as  
 9336       macros:

```

9337         LC_ALL
9338         LC_COLLATE
9339         LC_CTYPE
9340 CX      LC_MESSAGES
9341         LC_MONETARY
9342         LC_NUMERIC
9343         LC_TIME
  
```

9344       which shall expand to distinct integer constant expressions, for use as the first argument to the  
 9345       *setlocale()* function.

9346       Additional macro definitions, beginning with the characters *LC\_* and an uppercase letter, may  
 9347       also be given here.

9348           The following shall be declared as functions and may also be defined as macros. Function  
9349           prototypes shall be provided.

9350           struct lconv \*localeconv(void);  
9351           char    \*setlocale(int, const char \*);

#### 9352 **APPLICATION USAGE**

9353           None.

#### 9354 **RATIONALE**

9355           None.

#### 9356 **FUTURE DIRECTIONS**

9357           None.

#### 9358 **SEE ALSO**

9359           The System Interfaces volume of IEEE Std 1003.1-2001, *localeconv()*, *setlocale()*, Chapter 8 (on  
9360           page 161)

#### 9361 **CHANGE HISTORY**

9362           First released in Issue 3.

9363           Included for alignment with the ISO C standard.

#### 9364 **Issue 6**

9365           The **lconv** structure is expanded with new members (**int\_n\_cs\_precedes**, **int\_n\_sep\_by\_space**,  
9366           **int\_n\_sign\_posn**, **int\_p\_cs\_precedes**, **int\_p\_sep\_by\_space**, and **int\_p\_sign\_posn**) for alignment  
9367           with the ISO/IEC 9899:1999 standard.

9368           Extensions beyond the ISO C standard are marked.

9369 **NAME**9370 **math.h** — mathematical declarations9371 **SYNOPSIS**9372 `#include <math.h>`9373 **DESCRIPTION**

9374 **CX** Some of the functionality described on this reference page extends the ISO C standard.  
 9375 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 9376 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
 9377 symbols in this header.

9378 The <**math.h**> header shall include definitions for at least the following types:

9379 **float\_t** A real-floating type at least as wide as **float**.

9380 **double\_t** A real-floating type at least as wide as **double**, and at least as wide as **float\_t**.

9381 If FLT\_EVAL\_METHOD equals 0, **float\_t** and **double\_t** shall be **float** and **double**, respectively; if  
 9382 FLT\_EVAL\_METHOD equals 1, they shall both be **double**; if FLT\_EVAL\_METHOD equals 2,  
 9383 they shall both be **long double**; for other values of FLT\_EVAL\_METHOD, they are otherwise  
 9384 implementation-defined.

9385 The <**math.h**> header shall define the following macros, where real-floating indicates that the  
 9386 argument shall be an expression of real-floating type:

```
9387 int fpclassify(real-floating x);
9388 int isfinite(real-floating x);
9389 int isinf(real-floating x);
9390 int isnan(real-floating x);
9391 int isnormal(real-floating x);
9392 int signbit(real-floating x);
9393 int isgreater(real-floating x, real-floating y);
9394 int isgreaterequal(real-floating x, real-floating y);
9395 int isless(real-floating x, real-floating y);
9396 int islessequal(real-floating x, real-floating y);
9397 int islessgreater(real-floating x, real-floating y);
9398 int unordered(real-floating x, real-floating y);
```

9399 The <**math.h**> header shall provide for the following constants. The values are of type **double**  
 9400 and are accurate within the precision of the **double** type.

9401	<b>XSI</b>	<b>M_E</b>	Value of $e$
9402		<b>M_LOG2E</b>	Value of $\log_2 e$
9403		<b>M_LOG10E</b>	Value of $\log_{10} e$
9404		<b>M_LN2</b>	Value of $\log_e 2$
9405		<b>M_LN10</b>	Value of $\log_e 10$
9406		<b>M_PI</b>	Value of $\pi$
9407		<b>M_PI_2</b>	Value of $\pi/2$
9408		<b>M_PI_4</b>	Value of $\pi/4$
9409		<b>M_1_PI</b>	Value of $1/\pi$
9410		<b>M_2_PI</b>	Value of $2/\pi$



9411	M_2_SQRTPI	Value of $2\sqrt{\pi}$
9412	M_SQRT2	Value of $\sqrt{2}$
9413	M_SQRT1_2	Value of $1/\sqrt{2}$
9414	The header shall define the following symbolic constants:	
9415	XSI MAXFLOAT	Value of maximum non-infinite single-precision floating-point number.
9416	HUGE_VAL	A positive <b>double</b> expression, not necessarily representable as a <b>float</b> . Used as an error value returned by the mathematics library. HUGE_VAL evaluates to +infinity on systems supporting IEEE Std 754-1985.
9417		
9418		
9419	HUGE_VALF	A positive <b>float</b> constant expression. Used as an error value returned by the mathematics library. HUGE_VALF evaluates to +infinity on systems supporting IEEE Std 754-1985.
9420		
9421		
9422	HUGE_VALL	A positive <b>long double</b> constant expression. Used as an error value returned by the mathematics library. HUGE_VALL evaluates to +infinity on systems supporting IEEE Std 754-1985.
9423		
9424		
9425	INFINITY	A constant expression of type <b>float</b> representing positive or unsigned infinity, if available; else a positive constant of type <b>float</b> that overflows at translation time.
9426		
9427		
9428	NAN	A constant expression of type <b>float</b> representing a quiet NaN. This symbolic constant is only defined if the implementation supports quiet NaNs for the <b>float</b> type.
9429		
9430		
9431	The following macros shall be defined for number classification. They represent the mutually-exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with FP_ and an uppercase letter, may also be specified by the implementation.	
9432		
9433		
9434		
9435		
9436	FP_INFINITE	
9437	FP_NAN	
9438	FP_NORMAL	
9439	FP_SUBNORMAL	
9440	FP_ZERO	
9441	The following optional macros indicate whether the <i>fma()</i> family of functions are fast compared with direct code:	
9442		
9443	FP_FAST_FMA	
9444	FP_FAST_FMAF	
9445	FP_FAST_FMAL	
9446	If defined, the FP_FAST_FMA macro shall indicate that the <i>fma()</i> function generally executes about as fast as, or faster than, a multiply and an add of <b>double</b> operands. If undefined, the speed of execution is unspecified. The other macros have the equivalent meaning for the <b>float</b> and <b>long double</b> versions.	
9447		
9448		
9449		
9450	The following macros shall expand to integer constant expressions whose values are returned by <i>ilogb(x)</i> if <i>x</i> is zero or NaN, respectively. The value of FP_ILOGB0 shall be either {INT_MIN} or -{INT_MAX}. The value of FP_ILOGBNAN shall be either {INT_MAX} or {INT_MIN}.	
9451		
9452		

9453           FP\_ILOGB0  
 9454           FP\_ILOGBNAN

9455           The following macros shall expand to the integer constants 1 and 2, respectively;

9456           MATH\_ERRNO  
 9457           MATH\_ERREXCEPT

9458           The following macro shall expand to an expression that has type **int** and the value  
 9459           MATH\_ERRNO, MATH\_ERREXCEPT, or the bitwise-inclusive OR of both:

9460           math\_errhandling

9461           The value of math\_errhandling is constant for the duration of the program. It is unspecified  
 9462           whether math\_errhandling is a macro or an identifier with external linkage. If a macro definition  
 9463           is suppressed or a program defines an identifier with the name math\_errhandling, the behavior  
 9464           is undefined. If the expression (math\_errhandling & MATH\_ERREXCEPT) can be non-zero, the  
 9465           implementation shall define the macros FE\_DIVBYZERO, FE\_INVALID, and FE\_OVERFLOW in  
 9466           <fenv.h>.

9467           The following shall be declared as functions and may also be defined as macros. Function  
 9468           prototypes shall be provided.

9469           double        acos(double);  
 9470           float         acosf(float);  
 9471           double        acosh(double);  
 9472           float         acoshf(float);  
 9473           long double   acoshl(long double);  
 9474           long double   acosl(long double);  
 9475           double        asin(double);  
 9476           float         asinf(float);  
 9477           double        asinh(double);  
 9478           float         asinhf(float);  
 9479           long double   asinh1(long double);  
 9480           long double   asinl(long double);  
 9481           double        atan(double);  
 9482           double        atan2(double, double);  
 9483           float         atan2f(float, float);  
 9484           long double   atan2l(long double, long double);  
 9485           float         atanf(float);  
 9486           double        atanh(double);  
 9487           float         atanhf(float);  
 9488           long double   atanhl(long double);  
 9489           long double   atanl(long double);  
 9490           double        cbrt(double);  
 9491           float         cbrtf(float);  
 9492           long double   cbrtl(long double);  
 9493           double        ceil(double);  
 9494           float         ceilf(float);  
 9495           long double   ceill(long double);  
 9496           double        copysign(double, double);  
 9497           float         copysignf(float, float);  
 9498           long double   copysignl(long double, long double);  
 9499           double        cos(double);  
 9500           float         cosf(float);  
 9501           double        cosh(double);

```

9502     float      coshf(float);
9503     long double coshl(long double);
9504     long double cosl(long double);
9505     double      erf(double);
9506     double      erfc(double);
9507     float       erfcf(float);
9508     long double erfcl(long double);
9509     float       erff(float);
9510     long double erfl(long double);
9511     double      exp(double);
9512     double      exp2(double);
9513     float       exp2f(float);
9514     long double exp2l(long double);
9515     float       expf(float);
9516     long double expl(long double);
9517     double      expml(double);
9518     float       expmlf(float);
9519     long double expmll(long double);
9520     double      fabs(double);
9521     float       fabsf(float);
9522     long double fabsl(long double);
9523     double      fdim(double, double);
9524     float       fdimf(float, float);
9525     long double fdiml(long double, long double);
9526     double      floor(double);
9527     float       floorf(float);
9528     long double floorl(long double);
9529     double      fma(double, double, double);
9530     float       fmaf(float, float, float);
9531     long double fmal(long double, long double, long double);
9532     double      fmax(double, double);
9533     float       fmaxf(float, float);
9534     long double fmaxl(long double, long double);
9535     double      fmin(double, double);
9536     float       fminf(float, float);
9537     long double fminl(long double, long double);
9538     double      fmod(double, double);
9539     float       fmodf(float, float);
9540     long double fmodl(long double, long double);
9541     double      frexp(double, int *);
9542     float       frexpf(float value, int *);
9543     long double frexpl(long double value, int *);
9544     double      hypot(double, double);
9545     float       hypotf(float, float);
9546     long double hypotl(long double, long double);
9547     int         ilogb(double);
9548     int         ilogbf(float);
9549     int         ilogbl(long double);
9550 XSI     double      j0(double);
9551         double      j1(double);
9552         double      jn(int, double);
9553         double      ldexp(double, int);

```

```

9554     float      ldexpf(float, int);
9555     long double ldexpl(long double, int);
9556     double      lgamma(double);
9557     float      lgammaf(float);
9558     long double lgammal(long double);
9559     long long   llrint(double);
9560     long long   llrintf(float);
9561     long long   llrintl(long double);
9562     long long   llround(double);
9563     long long   llroundf(float);
9564     long long   llroundl(long double);
9565     double      log(double);
9566     double      log10(double);
9567     float      log10f(float);
9568     long double log10l(long double);
9569     double      log1p(double);
9570     float      log1pf(float);
9571     long double log1pl(long double);
9572     double      log2(double);
9573     float      log2f(float);
9574     long double log2l(long double);
9575     double      logb(double);
9576     float      logbf(float);
9577     long double logbl(long double);
9578     float      logf(float);
9579     long double logl(long double);
9580     long        lrint(double);
9581     long        lrintf(float);
9582     long        lrintl(long double);
9583     long        lround(double);
9584     long        lroundf(float);
9585     long        lroundl(long double);
9586     double      modf(double, double *);
9587     float      modff(float, float *);
9588     long double modfl(long double, long double *);
9589     double      nan(const char *);
9590     float      nanf(const char *);
9591     long double nanl(const char *);
9592     double      nearbyint(double);
9593     float      nearbyintf(float);
9594     long double nearbyintl(long double);
9595     double      nextafter(double, double);
9596     float      nextafterf(float, float);
9597     long double nextafterl(long double, long double);
9598     double      nexttoward(double, long double);
9599     float      nexttowardf(float, long double);
9600     long double nexttowardl(long double, long double);
9601     double      pow(double, double);
9602     float      powf(float, float);
9603     long double powl(long double, long double);
9604     double      remainder(double, double);
9605     float      remainderf(float, float);

```

```

9606     long double remainderl(long double, long double);
9607     double      remquo(double, double, int *);
9608     float       remquof(float, float, int *);
9609     long double remquol(long double, long double, int *);
9610     double      rint(double);
9611     float       rintf(float);
9612     long double rintl(long double);
9613     double      round(double);
9614     float       roundf(float);
9615     long double roundl(long double);
9616 XSI     double      scalb(double, double);
9617     double      scalbln(double, long);
9618     float       scalblnf(float, long);
9619     long double scalblnl(long double, long);
9620     double      scalbn(double, int);
9621     float       scalbnf(float, int);
9622     long double scalbnl(long double, int);
9623     double      sin(double);
9624     float       sinf(float);
9625     double      sinh(double);
9626     float       sinhf(float);
9627     long double sinhl(long double);
9628     long double sinl(long double);
9629     double      sqrt(double);
9630     float       sqrtf(float);
9631     long double sqrtl(long double);
9632     double      tan(double);
9633     float       tanf(float);
9634     double      tanh(double);
9635     float       tanhf(float);
9636     long double tanhl(long double);
9637     long double tanl(long double);
9638     double      tgamma(double);
9639     float       tgammaf(float);
9640     long double tgammal(long double);
9641     double      trunc(double);
9642     float       truncf(float);
9643     long double truncf(long double);
9644 XSI     double      y0(double);
9645     double      y1(double);
9646     double      yn(int, double);
9647

```

9648 The following external variable shall be defined:

```

9649 XSI     extern int signgam;
9650

```

9651 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces  
9652 volume of IEEE Std 1003.1-2001 for all representable values of its input arguments, except where  
9653 stated otherwise. Each function shall execute as if it were a single operation without generating  
9654 any externally visible exceptional conditions.

9655 **APPLICATION USAGE**

9656 The FP\_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is  
 9657 off) the implementation to contract expressions. Each pragma can occur either outside external  
 9658 declarations or preceding all explicit declarations and statements inside a compound statement.  
 9659 When outside external declarations, the pragma takes effect from its occurrence until another  
 9660 FP\_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a  
 9661 compound statement, the pragma takes effect from its occurrence until another FP\_CONTRACT  
 9662 pragma is encountered (including within a nested compound statement), or until the end of the  
 9663 compound statement; at the end of a compound statement the state for the pragma is restored to  
 9664 its condition just before the compound statement. If this pragma is used in any other context, the  
 9665 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

9666 **RATIONALE**

9667 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type  
 9668 **double**. All the names formed by appending 'f' or 'l' to a name in <math.h> were reserved  
 9669 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999  
 9670 standard provides for all three versions of math functions.

9671 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their  
 9672 capability is available through *sprintf()*. These are provided on XSI-conformant systems  
 9673 supporting the Legacy Option Group.

9674 **FUTURE DIRECTIONS**

9675 None.

9676 **SEE ALSO**

9677 <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *acos()*, *acosh()*,  
 9678 *asin()*, *atan()*, *atan2()*, *cbrt()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *expm1()*, *fabs()*, *floor()*, *fmod()*,  
 9679 *frexp()*, *hypot()*, *ilogb()*, *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *log1p()*, *logb()*, *modf()*,  
 9680 *nextafter()*, *pow()*, *remainder()*, *rint()*, *scalb()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*

9681 **CHANGE HISTORY**

9682 First released in Issue 1.

9683 **Issue 6**

9684 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

9685 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/21 is applied, making it clear that the 2  
 9686 meaning of the FP\_FAST\_FMA macro is unspecified if the macro is undefined. 2

9687 **NAME**

9688       monetary.h — monetary types

9689 **SYNOPSIS**

9690 XSI       #include <monetary.h>

9691

9692 **DESCRIPTION**

9693       The <monetary.h> header shall define the following types:

9694       **size\_t**               As described in <stddef.h>.

9695       **ssize\_t**             As described in <sys/types.h>.

9696       The following shall be declared as a function and may also be defined as a macro. A function  
9697       prototype shall be provided.

9698       **ssize\_t**   strfmon(char \*restrict, size\_t, const char \*restrict, ...);

9699 **APPLICATION USAGE**

9700       None.

9701 **RATIONALE**

9702       None.

9703 **FUTURE DIRECTIONS**

9704       None.

9705 **SEE ALSO**

9706       The System Interfaces volume of IEEE Std 1003.1-2001, *strfmon()*

9707 **CHANGE HISTORY**

9708       First released in Issue 4.

9709 **Issue 6**

9710       The **restrict** keyword is added to the prototype for *strfmon()*.

## 9711 NAME

9712 mqqueue.h — message queues (**REALTIME**)

## 9713 SYNOPSIS

9714 MSG #include &lt;mqqueue.h&gt;

9715

## 9716 DESCRIPTION

9717 The <mqqueue.h> header shall define the **mqd\_t** type, which is used for message queue  
9718 descriptors. This is not an array type.9719 The <mqqueue.h> header shall define the **sigevent** structure (as described in <signal.h>) and the  
9720 **mq\_attr** structure, which is used in getting and setting the attributes of a message queue.  
9721 Attributes are initially set when the message queue is created. An **mq\_attr** structure shall have at  
9722 least the following fields:

9723	long	mq_flags	Message queue flags.
9724	long	mq_maxmsg	Maximum number of messages.
9725	long	mq_msgsize	Maximum message size.
9726	long	mq_curmsgs	Number of messages currently queued.

9727 The following shall be declared as functions and may also be defined as macros. Function  
9728 prototypes shall be provided.

9729	int	mq_close(mqd_t);
9730	int	mq_getattr(mqd_t, struct mq_attr *);
9731	int	mq_notify(mqd_t, const struct sigevent *);
9732	mqd_t	mq_open(const char *, int, ...);
9733	ssize_t	mq_receive(mqd_t, char *, size_t, unsigned *);
9734	int	mq_send(mqd_t, const char *, size_t, unsigned);
9735	int	mq_setattr(mqd_t, const struct mq_attr *restrict,
9736		struct mq_attr *restrict);
9737	TMO	ssize_t mq_timedreceive(mqd_t, char *restrict, size_t,
9738		unsigned *restrict, const struct timespec *restrict);
9739	int	mq_timedsend(mqd_t, const char *, size_t, unsigned,
9740		const struct timespec *);
9741	int	mq_unlink(const char *);

9742 Inclusion of the <mqqueue.h> header may make visible symbols defined in the headers <fcntl.h>,  
9743 <signal.h>, <sys/types.h>, and <time.h>.

## 9744 APPLICATION USAGE

9745 None.

## 9746 RATIONALE

9747 None.

## 9748 FUTURE DIRECTIONS

9749 None.

## 9750 SEE ALSO

9751 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of  
9752 IEEE Std 1003.1-2001, **mq\_close()**, **mq\_getattr()**, **mq\_notify()**, **mq\_open()**, **mq\_receive()**, **mq\_send()**,  
9753 **mq\_setattr()**, **mq\_timedreceive()**, **mq\_timedsend()**, **mq\_unlink()**



9754 **CHANGE HISTORY**

9755 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

9756 **Issue 6**

9757 The <mqqueue.h> header is marked as part of the Message Passing option.

9758 The *mq\_timedreceive()* and *mq\_timedsend()* functions are added for alignment with  
9759 IEEE Std 1003.1d-1999.

9760 The **restrict** keyword is added to the prototypes for *mq\_setattr()* and *mq\_timedreceive()*.

9761 **NAME**

9762 ndbm.h — definitions for ndbm database operations

9763 **SYNOPSIS**9764 XSI `#include <ndbm.h>`

9765

9766 **DESCRIPTION**9767 The <ndbm.h> header shall define the **datum** type as a structure that includes at least the  
9768 following members:9769 `void *dptr` A pointer to the application's data.9770 `size_t dsize` The size of the object pointed to by *dptr*.9771 The `size_t` type shall be defined as described in <stddef.h>.9772 The <ndbm.h> header shall define the **DBM** type.9773 The following constants shall be defined as possible values for the *store\_mode* argument to  
9774 *dbm\_store()*:9775 `DBM_INSERT` Insertion of new entries only.9776 `DBM_REPLACE` Allow replacing existing entries.9777 The following shall be declared as functions and may also be defined as macros. Function  
9778 prototypes shall be provided.9779 `int dbm_clearerr(DBM *);`9780 `void dbm_close(DBM *);`9781 `int dbm_delete(DBM *, datum);`9782 `int dbm_error(DBM *);`9783 `datum dbm_fetch(DBM *, datum);`9784 `datum dbm_firstkey(DBM *);`9785 `datum dbm_nextkey(DBM *);`9786 `DBM *dbm_open(const char *, int, mode_t);`9787 `int dbm_store(DBM *, datum, datum, int);`9788 The `mode_t` type shall be defined through **typedef** as described in <sys/types.h>.9789 **APPLICATION USAGE**

9790 None.

9791 **RATIONALE**

9792 None.

9793 **FUTURE DIRECTIONS**

9794 None.

9795 **SEE ALSO**9796 <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *dbm\_clearerr()*9797 **CHANGE HISTORY**

9798 First released in Issue 4, Version 2.

9799 **Issue 5**9800 References to the definitions of **size\_t** and **mode\_t** are added to the DESCRIPTION.

9801 **NAME**

9802 net/if.h — sockets local interfaces

9803 **SYNOPSIS**

9804 #include <net/if.h>

9805 **DESCRIPTION**

9806 The <net/if.h> header shall define the **if\_nameindex** structure that includes at least the  
9807 following members:

9808 unsigned if\_index    Numeric index of the interface.  
9809 char        \*if\_name    Null-terminated name of the interface.

9810 The <net/if.h> header shall define the following macro for the length of a buffer containing an  
9811 interface name (including the terminating NULL character):

9812 IF\_NAMESIZE    Interface name length.

9813 The following shall be declared as functions and may also be defined as macros. Function  
9814 prototypes shall be provided.

9815 unsigned                if\_nametoindex(const char \*);  
9816 char                    \*if\_indextoname(unsigned, char \*);  
9817 struct if\_nameindex    \*if\_nameindex(void);  
9818 void                    if\_freenameindex(struct if\_nameindex \*);

9819 **APPLICATION USAGE**

9820 None.

9821 **RATIONALE**

9822 None.

9823 **FUTURE DIRECTIONS**

9824 None.

9825 **SEE ALSO**

9826 The System Interfaces volume of IEEE Std 1003.1-2001, *if\_freenameindex()*, *if\_indextoname()*,  
9827 *if\_nameindex()*, *if\_nametoindex()*

9828 **CHANGE HISTORY**

9829 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9830 **NAME**

9831 netdb.h — definitions for network database operations

9832 **SYNOPSIS**

9833 #include &lt;netdb.h&gt;

9834 **DESCRIPTION**9835 The <netdb.h> header may define the **in\_port\_t** type and the **in\_addr\_t** type as described in  
9836 <netinet/in.h>.9837 The <netdb.h> header shall define the **hostent** structure that includes at least the following  
9838 members:

9839	char	*h_name	Official name of the host.
9840	char	**h_aliases	A pointer to an array of pointers to
9841			alternative host names, terminated by a
9842			null pointer.
9843	int	h_addrtype	Address type.
9844	int	h_length	The length, in bytes, of the address.
9845	char	**h_addr_list	A pointer to an array of pointers to network
9846			addresses (in network byte order) for the host,
9847			terminated by a null pointer.

9848 The <netdb.h> header shall define the **netent** structure that includes at least the following  
9849 members:

9850	char	*n_name	Official, fully-qualified (including the
9851			domain) name of the host.
9852	char	**n_aliases	A pointer to an array of pointers to
9853			alternative network names, terminated by a
9854			null pointer.
9855	int	n_addrtype	The address type of the network.
9856	uint32_t	n_net	The network number, in host byte order.

9857 The **uint32\_t** type shall be defined as described in <inttypes.h>.9858 The <netdb.h> header shall define the **protoent** structure that includes at least the following  
9859 members:

9860	char	*p_name	Official name of the protocol.
9861	char	**p_aliases	A pointer to an array of pointers to
9862			alternative protocol names, terminated by
9863			a null pointer.
9864	int	p_proto	The protocol number.

9865 The <netdb.h> header shall define the **servent** structure that includes at least the following  
9866 members:

9867	char	*s_name	Official name of the service.
9868	char	**s_aliases	A pointer to an array of pointers to
9869			alternative service names, terminated by
9870			a null pointer.
9871	int	s_port	The port number at which the service
9872			resides, in network byte order.
9873	char	*s_proto	The name of the protocol to use when
9874			contacting the service.

9875 The <netdb.h> header shall define the IPPORT\_RESERVED macro with the value of the highest  
9876 reserved Internet port number.

9877 OB When the <netdb.h> header is included, *h\_errno* shall be available as a modifiable lvalue of type  
9878 **int**. It is unspecified whether *h\_errno* is a macro or an identifier declared with external linkage.

9879 The <netdb.h> header shall define the following macros for use as error values for  
9880 *gethostbyaddr()* and *gethostbyname()*:

9881 HOST\_NOT\_FOUND  
9882 NO\_DATA  
9883 NO\_RECOVERY  
9884 TRY\_AGAIN

## 9885 Address Information Structure

9886 The <netdb.h> header shall define the **addrinfo** structure that includes at least the following  
9887 members:

9888	int	ai_flags	Input flags.
9889	int	ai_family	Address family of socket.
9890	int	ai_socktype	Socket type.
9891	int	ai_protocol	Protocol of socket.
9892	socklen_t	ai_addrlen	Length of socket address.
9893	struct sockaddr	*ai_addr	Socket address of socket.
9894	char	*ai_canonname	Canonical name of service location.
9895	struct addrinfo	*ai_next	Pointer to next in list.

9896 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer  
9897 constants for use in the *flags* field of the **addrinfo** structure:

9898 AI\_PASSIVE Socket address is intended for *bind()*.

9899 AI\_CANONNAME  
9900 Request for canonical name.

9901 AI\_NUMERICHOST  
9902 Return numeric host address as name.

9903 AI\_NUMERICSERV  
9904 Inhibit service name resolution.

9905 AI\_V4MAPPED If no IPv6 addresses are found, query for IPv4 addresses and return them to  
9906 the caller as IPv4-mapped IPv6 addresses.

9907 AI\_ALL Query for both IPv4 and IPv6 addresses.

9908 AI\_ADDRCONFIG  
9909 Query for IPv4 addresses only when an IPv4 address is configured; query for  
9910 IPv6 addresses only when an IPv6 address is configured.

9911 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer  
9912 constants for use in the *flags* argument to *getnameinfo()*:

9913 NI\_NOFQDN Only the nodename portion of the FQDN is returned for local hosts.

9914 NI\_NUMERICHOST  
9915 The numeric form of the node's address is returned instead of its name.

9916 NI\_NAMEREQD Return an error if the node's name cannot be located in the database.

9917 NI\_NUMERICSERV

9918 The numeric form of the service address is returned instead of its name.

9919 NI\_NUMERICSCOPE 1

9920 For IPv6 addresses, the numeric form of the scope identifier is returned 1

9921 instead of its name. 1

9922 NI\_DGRAM Indicates that the service is a datagram service (SOCK\_DGRAM).

9923 **Address Information Errors**

9924 The <netdb.h> header shall define the following macros for use as error values for *getaddrinfo()*

9925 and *getnameinfo()*:

9926 EAI\_AGAIN The name could not be resolved at this time. Future attempts may succeed.

9927 EAI\_BADFLAGS The flags had an invalid value.

9928 EAI\_FAIL A non-recoverable error occurred.

9929 EAI\_FAMILY The address family was not recognized or the address length was invalid for

9930 the specified family.

9931 EAI\_MEMORY There was a memory allocation failure.

9932 EAI\_NONAME The name does not resolve for the supplied parameters.

9933 NI\_NAMEREQD is set and the host's name cannot be located, or both

9934 *nodename* and *servname* were null.

9935 EAI\_SERVICE The service passed was not recognized for the specified socket type.

9936 EAI\_SOCKETTYPE The intended socket type was not recognized.

9937 EAI\_SYSTEM A system error occurred. The error code can be found in *errno*.

9938 EAI\_OVERFLOW

9939 An argument buffer overflowed.

9940 The following shall be declared as functions and may also be defined as macros. Function

9941 prototypes shall be provided.

9942 void endhostent(void);

9943 void endnetent(void);

9944 void endprotoent(void);

9945 void endservent(void);

9946 void freeaddrinfo(struct addrinfo \*);

9947 const char \*gai\_strerror(int);

9948 int getaddrinfo(const char \*restrict, const char \*restrict,

9949 const struct addrinfo \*restrict,

9950 struct addrinfo \*\*restrict);

9951 struct hostent \*gethostbyaddr(const void \*, socklen\_t, int);

9952 struct hostent \*gethostbyname(const char \*);

9953 struct hostent \*gethostent(void);

9954 int getnameinfo(const struct sockaddr \*restrict, socklen\_t,

9955 char \*restrict, socklen\_t, char \*restrict,

9956 socklen\_t, int); 1

9957 struct netent \*getnetbyaddr(uint32\_t, int);

9958 struct netent \*getnetbyname(const char \*);

```

9959      struct netent      *getnetent(void);
9960      struct protoent    *getprotobyname(const char *);
9961      struct protoent    *getprotobynumber(int);
9962      struct protoent    *getprotoent(void);
9963      struct servent     *getservbyname(const char *, const char *);
9964      struct servent     *getservbyport(int, const char *);
9965      struct servent     *getservent(void);
9966      void               sethostent(int);
9967      void               setnetent(int);
9968      void               setprotoent(int);
9969      void               setservent(int);

```

9970 The type **socklen\_t** shall be defined through **typedef** as described in <sys/socket.h>.

9971 Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>,  
9972 <sys/socket.h>, and <inttypes.h>.

#### 9973 APPLICATION USAGE

9974 None.

#### 9975 RATIONALE

9976 None.

#### 9977 FUTURE DIRECTIONS

9978 None.

#### 9979 SEE ALSO

9980 <netinet/in.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of  
9981 IEEE Std 1003.1-2001, *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *getaddrinfo()*,  
9982 *getnameinfo()*

#### 9983 CHANGE HISTORY

9984 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9985 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for  
9986 *gai\_strerror()* from **char \*** to **const char \***. This is for coordination with the IPnG Working Group.

9987 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the 1  
9988 NI\_NUMERICSCOPE macro and correcting the *getnameinfo()* function prototype. These changes 1  
9989 are for alignment with IPv6. 1

## 9990 NAME

9991       netinet/in.h — Internet address family

## 9992 SYNOPSIS

9993       #include &lt;netinet/in.h&gt;

## 9994 DESCRIPTION

9995       The &lt;netinet/in.h&gt; header shall define the following types:

9996       **in\_port\_t**     Equivalent to the type **uint16\_t** as defined in <inttypes.h>.9997       **in\_addr\_t**     Equivalent to the type **uint32\_t** as defined in <inttypes.h>.9998       The **sa\_family\_t** type shall be defined as described in <sys/socket.h>.9999       The **uint8\_t** and **uint32\_t** type shall be defined as described in <inttypes.h>. Inclusion of the  
10000       <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.10001       The <netinet/in.h> header shall define the **in\_addr** structure that includes at least the following  
10002       member:

10003       in\_addr\_t   s\_addr

10004       The <netinet/in.h> header shall define the **sockaddr\_in** structure that includes at least the 2  
10005       following members: 210006       sa\_family\_t       sin\_family     AF\_INET.  
10007       in\_port\_t         sin\_port       Port number.  
10008       struct in\_addr   sin\_addr       IP address.10009       The *sin\_port* and *sin\_addr* members shall be in network byte order. 210010       The **sockaddr\_in** structure is used to store addresses for the Internet address family. Values of  
10011       this type shall be cast by applications to **struct sockaddr** for use with socket functions.10012 IP6       The <netinet/in.h> header shall define the **in6\_addr** structure that contains at least the following  
10013       member:

10014       uint8\_t s6\_addr[16]

10015       This array is used to contain a 128-bit IPv6 address, stored in network byte order.

10016       The <netinet/in.h> header shall define the **sockaddr\_in6** structure that includes at least the 2  
10017       following members: 210018       sa\_family\_t       sin6\_family    AF\_INET6.  
10019       in\_port\_t         sin6\_port       Port number.  
10020       uint32\_t          sin6\_flowinfo   IPv6 traffic class and flow information.  
10021       struct in6\_addr   sin6\_addr       IPv6 address.  
10022       uint32\_t          sin6\_scope\_id   Set of interfaces for a scope.10023       The *sin6\_port* and *sin6\_addr* members shall be in network byte order. 210024       The **sockaddr\_in6** structure shall be set to zero by an application prior to using it, since  
10025       implementations are free to have additional, implementation-defined fields in **sockaddr\_in6**.10026       The *sin6\_scope\_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the  
10027       scope of the address carried in the *sin6\_addr* field. For a link scope *sin6\_addr*, the application  
10028       shall ensure that *sin6\_scope\_id* is a link index. For a site scope *sin6\_addr*, the application shall  
10029       ensure that *sin6\_scope\_id* is a site index. The mapping of *sin6\_scope\_id* to an interface or set of  
10030       interfaces is implementation-defined.



10031	The <netinet/in.h> header shall declare the following external variable:	
10032	const struct in6_addr in6addr_any	
10033	This variable is initialized by the system to contain the wildcard IPv6 address. The	
10034	<netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be	
10035	constant at compile time and can be used to initialize a variable of type <b>struct in6_addr</b> to the	
10036	IPv6 wildcard address.	
10037	The <netinet/in.h> header shall declare the following external variable:	
10038	const struct in6_addr in6addr_loopback	
10039	This variable is initialized by the system to contain the loopback IPv6 address. The	
10040	<netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be	
10041	constant at compile time and can be used to initialize a variable of type <b>struct in6_addr</b> to the	
10042	IPv6 loopback address.	
10043	The <netinet/in.h> header shall define the <b>ipv6_mreq</b> structure that includes at least the	
10044	following members:	
10045	struct in6_addr	ipv6mr_multiaddr IPv6 multicast address.
10046	unsigned	ipv6mr_interface Interface index.
10047		
10048	The <netinet/in.h> header shall define the following macros for use as values of the <i>level</i>	
10049	argument of <i>getsockopt()</i> and <i>setsockopt()</i> :	
10050	IPPROTO_IP	Internet protocol.
10051	IPPROTO_IPV6	Internet Protocol Version 6.
10052	IPPROTO_ICMP	Control message protocol.
10053	IPPROTO_RAW	Raw IP Packets Protocol.
10054	IPPROTO_TCP	Transmission control protocol.
10055	IPPROTO_UDP	User datagram protocol.
10056	The <netinet/in.h> header shall define the following macros for use as destination addresses for	
10057	<i>connect()</i> , <i>sendmsg()</i> , and <i>sendto()</i> :	
10058	INADDR_ANY	IPv4 local host address.
10059	INADDR_BROADCAST	IPv4 broadcast address.
10060	The <netinet/in.h> header shall define the following macro to help applications declare buffers	
10061	of the proper size to store IPv4 addresses in string form:	
10062	INET_ADDRSTRLEN	16. Length of the string form for IP.
10063	The <i>htonl()</i> , <i>htons()</i> , <i>ntohl()</i> , and <i>ntohs()</i> functions shall be available as defined in <arpa/inet.h>.	
10064	Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.	
10065	The <netinet/in.h> header shall define the following macro to help applications declare buffers	
10066	of the proper size to store IPv6 addresses in string form:	
10067	INET6_ADDRSTRLEN	46. Length of the string form for IPv6.
10068	The <netinet/in.h> header shall define the following macros, with distinct integer values, for use	
10069	in the <i>option_name</i> argument in the <i>getsockopt()</i> or <i>setsockopt()</i> functions at protocol level	
10070	IPPROTO_IPV6:	

10071	IPV6_JOIN_GROUP	Join a multicast group.
10072	IPV6_LEAVE_GROUP	Quit a multicast group.
10073	IPV6_MULTICAST_HOPS	
10074		Multicast hop limit.
10075	IPV6_MULTICAST_IF	Interface to use for outgoing multicast packets.
10076	IPV6_MULTICAST_LOOP	
10077		Multicast packets are delivered back to the local application.
10078	IPV6_UNICAST_HOPS	Unicast hop limit.
10079	IPV6_V6ONLY	Restrict AF_INET6 socket to IPv6 communications only.
10080	The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.	
10081	Each macro is of type <b>int</b> and takes a single argument of type <b>const struct in6_addr *</b> :	
10082	IN6_IS_ADDR_UNSPECIFIED	
10083		Unspecified address.
10084	IN6_IS_ADDR_LOOPBACK	
10085		Loopback address.
10086	IN6_IS_ADDR_MULTICAST	
10087		Multicast address.
10088	IN6_IS_ADDR_LINKLOCAL	
10089		Unicast link-local address.
10090	IN6_IS_ADDR_SITELOCAL	
10091		Unicast site-local address.
10092	IN6_IS_ADDR_V4MAPPED	
10093		IPv4 mapped address.
10094	IN6_IS_ADDR_V4COMPAT	
10095		IPv4-compatible address.
10096	IN6_IS_ADDR_MC_NODELOCAL	
10097		Multicast node-local address.
10098	IN6_IS_ADDR_MC_LINKLOCAL	
10099		Multicast link-local address.
10100	IN6_IS_ADDR_MC_SITELOCAL	
10101		Multicast site-local address.
10102	IN6_IS_ADDR_MC_ORGLOCAL	
10103		Multicast organization-local address.
10104	IN6_IS_ADDR_MC_GLOBAL	
10105		Multicast global address.

10106 **APPLICATION USAGE**

10107       None.

10108 **RATIONALE**

10109       None.

10110 **FUTURE DIRECTIONS**

10111       None.

10112 **SEE ALSO**

10113       Section 4.8 (on page 99), <arpa/inet.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces  
10114       volume of IEEE Std 1003.1-2001, *connect()*, *getsockopt()*, *htonl()*, *htons()*, *ntohl()*, *ntohs()*,  
10115       *sendmsg()*, *sendto()*, *setsockopt()*

10116 **CHANGE HISTORY**

10117       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10118       The *sin\_zero* member was removed from the **sockaddr\_in** structure as per The Open Group Base  
10119       Resolution bwg2001-004.

10120       IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to 1  
10121       the *in6addr\_any* and *in6addr\_loopback* external variables. 1

10122       IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/22 is applied, making it clear which 2  
10123       structure members are in network byte order. 2

**10124 NAME**

10125       netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

**10126 SYNOPSIS**

10127       #include <netinet/tcp.h>

**10128 DESCRIPTION**

10129       The **<netinet/tcp.h>** header shall define the following macro for use as a socket option at the IPPROTO\_TCP level:

10131       TCP\_NODELAY   Avoid coalescing of small segments.

10132       The macro shall be defined in the header. The implementation need not allow the value of the option to be set via *setsockopt()* or retrieved via *getsockopt()*.

**10134 APPLICATION USAGE**

10135       None.

**10136 RATIONALE**

10137       None.

**10138 FUTURE DIRECTIONS**

10139       None.

**10140 SEE ALSO**

10141       **<sys/socket.h>**, the System Interfaces volume of IEEE Std 1003.1-2001, *getsockopt()*, *setsockopt()*

**10142 CHANGE HISTORY**

10143       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10144 **NAME**

10145 nl\_types.h — data types

10146 **SYNOPSIS**

10147 XSI #include <nl\_types.h>

10148

10149 **DESCRIPTION**

10150 The <nl\_types.h> header shall contain definitions of at least the following types:

10151 **nl\_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*  
10152 to identify a catalog descriptor.

10153 **nl\_item** Used by *nl\_langinfo()* to identify items of *langinfo* data. Values of objects  
10154 of type **nl\_item** are defined in <langinfo.h>.

10155 The <nl\_types.h> header shall contain definitions of at least the following constants:

10156 **NL\_SETD** Used by *gencat* when no *\$set* directive is specified in a message text source  
10157 file; see the Internationalization Guide. This constant can be passed as the  
10158 value of *set\_id* on subsequent calls to *catgets()* (that is, to retrieve  
10159 messages from the default message set). The value of **NL\_SETD** is  
10160 implementation-defined.

10161 **NL\_CAT\_LOCALE** Value that must be passed as the *offlag* argument to *catopen()* to ensure  
10162 that message catalog selection depends on the *LC\_MESSAGES* locale  
10163 category, rather than directly on the *LANG* environment variable.

10164 The following shall be declared as functions and may also be defined as macros. Function  
10165 prototypes shall be provided.

```
10166 int      catclose(nl_catd);
10167 char     *catgets(nl_catd, int, int, const char *);
10168 nl_catd  catopen(const char *, int);
```

10169 **APPLICATION USAGE**

10170 None.

10171 **RATIONALE**

10172 None.

10173 **FUTURE DIRECTIONS**

10174 None.

10175 **SEE ALSO**

10176 <langinfo.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *catclose()*, *catgets()*,  
10177 *catopen()*, *nl\_langinfo()*, the Shell and Utilities volume of IEEE Std 1003.1-2001, *gencat*

10178 **CHANGE HISTORY**

10179 First released in Issue 2.

## 10180 NAME

10181 poll.h — definitions for the poll() function

## 10182 SYNOPSIS

10183 XSI #include &lt;poll.h&gt;

10184

## 10185 DESCRIPTION

10186 The <poll.h> header shall define the **pollfd** structure that includes at least the following  
10187 members:

10188 int fd The following descriptor being polled.

10189 short events The input event flags (see below).

10190 short revents The output event flags (see below).

10191 The <poll.h> header shall define the following type through **typedef**:10192 **nfds\_t** An unsigned integer type used for the number of file descriptors.10193 The implementation shall support one or more programming environments in which the width  
10194 of **nfds\_t** is no greater than the width of type **long**. The names of these programming  
10195 environments can be obtained using the *confstr()* function or the *getconf* utility.10196 The following symbolic constants shall be defined, zero or more of which may be OR'ed together  
10197 to form the *events* or *revents* members in the **pollfd** structure:

10198 POLLIN Data other than high-priority data may be read without blocking.

10199 POLLRDNORM Normal data may be read without blocking.

10200 POLLRDBAND Priority data may be read without blocking.

10201 POLLPRI High priority data may be read without blocking.

10202 POLLOUT Normal data may be written without blocking.

10203 POLLWRNORM Equivalent to POLLOUT.

10204 POLLWRBAND Priority data may be written.

10205 POLLERR An error has occurred (*revents* only).10206 POLLHUP Device has been disconnected (*revents* only).10207 POLLNVAL Invalid *fd* member (*revents* only).10208 The significance and semantics of normal, priority, and high-priority data are file and device-  
10209 specific.10210 The following shall be declared as a function and may also be defined as a macro. A function  
10211 prototype shall be provided.

10212 int poll(struct pollfd[], nfds\_t, int);

10213 **APPLICATION USAGE**

10214           None.

10215 **RATIONALE**

10216           None.

10217 **FUTURE DIRECTIONS**

10218           None.

10219 **SEE ALSO**

10220           The System Interfaces volume of IEEE Std 1003.1-2001, *confstr()*, *poll()*, the Shell and Utilities

10221           volume of IEEE Std 1003.1-2001, *getconf*

10222 **CHANGE HISTORY**

10223           First released in Issue 4, Version 2.

10224 **Issue 6**

10225           The description of the symbolic constants is updated to match the *poll()* function.

10226           Text related to STREAMS has been moved to the *poll()* reference page.

10227           A note is added to the DESCRIPTION regarding the significance and semantics of normal,  
10228           priority, and high-priority data.

10229 **NAME**

10230 pthread.h — threads

10231 **SYNOPSIS**

10232 THR #include &lt;pthread.h&gt;

10233

10234 **DESCRIPTION**

10235 The &lt;pthread.h&gt; header shall define the following symbols:

10236 BAR PTHREAD\_BARRIER\_SERIAL\_THREAD

10237 PTHREAD\_CANCEL\_ASYNCHRONOUS

10238 PTHREAD\_CANCEL\_ENABLE

10239 PTHREAD\_CANCEL\_DEFERRED

10240 PTHREAD\_CANCEL\_DISABLE

10241 PTHREAD\_CANCELED

10242 PTHREAD\_COND\_INITIALIZER

10243 PTHREAD\_CREATE\_DETACHED

10244 PTHREAD\_CREATE\_JOINABLE

10245 PTHREAD\_EXPLICIT\_SCHED

10246 PTHREAD\_INHERIT\_SCHED

10247 XSI PTHREAD\_MUTEX\_DEFAULT

10248 PTHREAD\_MUTEX\_ERRORCHECK

10249 PTHREAD\_MUTEX\_INITIALIZER

10250 XSI PTHREAD\_MUTEX\_NORMAL

10251 PTHREAD\_MUTEX\_RECURSIVE

10252 PTHREAD\_ONCE\_INIT

10253 TPI PTHREAD\_PRIO\_INHERIT 1

10254 TPP | TPI PTHREAD\_PRIO\_NONE 1

10255 TPP PTHREAD\_PRIO\_PROTECT 1

10256 PTHREAD\_PROCESS\_SHARED

10257 PTHREAD\_PROCESS\_PRIVATE

10258 TPS PTHREAD\_SCOPE\_PROCESS

10259 PTHREAD\_SCOPE\_SYSTEM

10260

10261 The following types shall be defined as described in &lt;sys/types.h&gt;:

10262 pthread\_attr\_t

10263 BAR pthread\_barrier\_t

10264 pthread\_barrierattr\_t

10265 pthread\_cond\_t

10266 pthread\_condattr\_t

10267 pthread\_key\_t

10268 pthread\_mutex\_t

10269 pthread\_mutexattr\_t

10270 pthread\_once\_t

10271 pthread\_rwlock\_t

10272 pthread\_rwlockattr\_t

10273 SPI pthread\_spinlock\_t

10274 pthread\_t

10275 The following shall be declared as functions and may also be defined as macros. Function  
 10276 prototypes shall be provided.



```

10277     int    pthread_atfork(void (*)(void), void (*)(void),
10278                          void (*)(void));
10279     int    pthread_attr_destroy(pthread_attr_t *);
10280     int    pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10281 XSI     int    pthread_attr_getguardsize(const pthread_attr_t *restrict,
10282                                         size_t *restrict);
10283 TPS     int    pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10284                                             int *restrict);
10285     int    pthread_attr_getschedparam(const pthread_attr_t *restrict,
10286                                       struct sched_param *restrict);
10287 TPS     int    pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10288                                           int *restrict);
10289 TPS     int    pthread_attr_getscope(const pthread_attr_t *restrict,
10290                                     int *restrict);
10291 TSA TSS int    pthread_attr_getstack(const pthread_attr_t *restrict,
10292                                     void **restrict, size_t *restrict);
10293 TSA     int    pthread_attr_getstackaddr(const pthread_attr_t *restrict,
10294                                         void **restrict);
10295 TSS     int    pthread_attr_getstacksize(const pthread_attr_t *restrict,
10296                                         size_t *restrict);
10297     int    pthread_attr_init(pthread_attr_t *);
10298     int    pthread_attr_setdetachstate(pthread_attr_t *, int);
10299 XSI     int    pthread_attr_setguardsize(pthread_attr_t *, size_t);
10300 TPS     int    pthread_attr_setinheritsched(pthread_attr_t *, int);
10301     int    pthread_attr_setschedparam(pthread_attr_t *restrict,
10302                                       const struct sched_param *restrict);
10303 TPS     int    pthread_attr_setschedpolicy(pthread_attr_t *, int);
10304     int    pthread_attr_setscope(pthread_attr_t *, int);
10305 TSA TSS int    pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10306 TSA     int    pthread_attr_setstackaddr(pthread_attr_t *, void *);
10307 TSS     int    pthread_attr_setstacksize(pthread_attr_t *, size_t);
10308 BAR     int    pthread_barrier_destroy(pthread_barrier_t *);
10309     int    pthread_barrier_init(pthread_barrier_t *restrict,
10310                                const pthread_barrierattr_t *restrict,
10311                                unsigned);
10311     int    pthread_barrier_wait(pthread_barrier_t *);
10312     int    pthread_barrierattr_destroy(pthread_barrierattr_t *);
10313 BAR TSH int    pthread_barrierattr_getpshared(
10314                                     const pthread_barrierattr_t *restrict, int *restrict);
10315 BAR     int    pthread_barrierattr_init(pthread_barrierattr_t *);
10316 BAR TSH int    pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10317     int    pthread_cancel(pthread_t);
10318     void    pthread_cleanup_push(void (*)(void *), void *);
10319     void    pthread_cleanup_pop(int);
10320     int    pthread_cond_broadcast(pthread_cond_t *);
10321     int    pthread_cond_destroy(pthread_cond_t *);
10322     int    pthread_cond_init(pthread_cond_t *restrict,
10323                             const pthread_condattr_t *restrict);
10324     int    pthread_cond_signal(pthread_cond_t *);
10325     int    pthread_cond_timedwait(pthread_cond_t *restrict,
10326                                  pthread_mutex_t *restrict, const struct timespec *restrict);
10327     int    pthread_cond_wait(pthread_cond_t *restrict,
10328                             pthread_mutex_t *restrict);

```

1

1

```

10329      int      pthread_condattr_destroy(pthread_condattr_t *);
10330 CS      int      pthread_condattr_getclock(const pthread_condattr_t *restrict,
10331      clockid_t *restrict);
10332 TSH      int      pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10333      int *restrict);
10334      int      pthread_condattr_init(pthread_condattr_t *);
10335 CS      int      pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10336 TSH      int      pthread_condattr_setpshared(pthread_condattr_t *, int);
10337      int      pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10338      void (*)(void *), void *restrict);
10339      int      pthread_detach(pthread_t);
10340      int      pthread_equal(pthread_t, pthread_t);
10341      void      pthread_exit(void *);
10342 XSI      int      pthread_getconcurrency(void);
10343 TCT      int      pthread_getcpuclockid(pthread_t, clockid_t *);
10344 TPS      int      pthread_getschedparam(pthread_t, int *restrict,
10345      struct sched_param *restrict);
10346      void *pthread_getspecific(pthread_key_t);
10347      int      pthread_join(pthread_t, void **);
10348      int      pthread_key_create(pthread_key_t *, void (*)(void *));
10349      int      pthread_key_delete(pthread_key_t);
10350      int      pthread_mutex_destroy(pthread_mutex_t *);
10351 TPP      int      pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10352      int *restrict);
10353      int      pthread_mutex_init(pthread_mutex_t *restrict,
10354      const pthread_mutexattr_t *restrict);
10355      int      pthread_mutex_lock(pthread_mutex_t *);
10356 TPP      int      pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10357      int *restrict);
10358 TMO      int      pthread_mutex_timedlock(pthread_mutex_t *,
10359      const struct timespec *);
10360      int      pthread_mutex_trylock(pthread_mutex_t *);
10361      int      pthread_mutex_unlock(pthread_mutex_t *);
10362      int      pthread_mutexattr_destroy(pthread_mutexattr_t *);
10363 TPP      int      pthread_mutexattr_getprioceiling(
10364      const pthread_mutexattr_t *restrict, int *restrict);
10365 TPP|TPI  int      pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10366      int *restrict);
10367 TSH      int      pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict, 1
10368      int *restrict);
10369 XSI      int      pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10370      int *restrict);
10371      int      pthread_mutexattr_init(pthread_mutexattr_t *);
10372 TPP      int      pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10373 TPP|TPI  int      pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10374 TSH      int      pthread_mutexattr_setpshared(pthread_mutexattr_t *, int); 1
10375 XSI      int      pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10376      int      pthread_once(pthread_once_t *, void (*)(void));
10377      int      pthread_rwlock_destroy(pthread_rwlock_t *);
10378      int      pthread_rwlock_init(pthread_rwlock_t *restrict,
10379      const pthread_rwlockattr_t *restrict);
10380      int      pthread_rwlock_rdlock(pthread_rwlock_t *);

```

```

10381 TMO      int      pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10382           const struct timespec *restrict);
10383      int      pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10384           const struct timespec *restrict);
10385      int      pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10386      int      pthread_rwlock_trywrlock(pthread_rwlock_t *);
10387      int      pthread_rwlock_unlock(pthread_rwlock_t *);
10388      int      pthread_rwlock_wrlock(pthread_rwlock_t *);
10389      int      pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10390 TSH      int      pthread_rwlockattr_getpshared(
10391           const pthread_rwlockattr_t *restrict, int *restrict);
10392      int      pthread_rwlockattr_init(pthread_rwlockattr_t *);
10393 TSH      int      pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10394 pthread_t
10395      pthread_self(void);
10396      int      pthread_setcancelstate(int, int *);
10397      int      pthread_setcanceltype(int, int *);
10398 XSI      int      pthread_setconcurrency(int);
10399 TPS      int      pthread_setschedparam(pthread_t, int,
10400           const struct sched_param *);
10401 TPS      int      pthread_setschedprio(pthread_t, int);
10402      int      pthread_setspecific(pthread_key_t, const void *);
10403 SPI      int      pthread_spin_destroy(pthread_spinlock_t *);
10404      int      pthread_spin_init(pthread_spinlock_t *, int);
10405      int      pthread_spin_lock(pthread_spinlock_t *);
10406      int      pthread_spin_trylock(pthread_spinlock_t *);
10407      int      pthread_spin_unlock(pthread_spinlock_t *);
10408      void     pthread_testcancel(void);

```

10409 Inclusion of the <pthread.h> header shall make symbols defined in the headers <sched.h> and  
10410 <time.h> visible.

## 10411 APPLICATION USAGE

10412 None.

## 10413 RATIONALE

10414 None.

## 10415 FUTURE DIRECTIONS

10416 None.

## 10417 SEE ALSO

10418 <sched.h>, <sys/types.h>, <time.h>, the System Interfaces volume of IEEE Std 1003.1-2001,  
10419 *pthread\_attr\_getguardsize()*, *pthread\_attr\_init()*, *pthread\_attr\_setscope()*, *pthread\_barrier\_destroy()*,  
10420 *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*, *pthread\_barrierattr\_destroy()*,  
10421 *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_init()*, *pthread\_barrierattr\_setpshared()*,  
10422 *pthread\_cancel()*, *pthread\_cleanup\_pop()*, *pthread\_cond\_init()*, *pthread\_cond\_signal()*,  
10423 *pthread\_cond\_wait()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_init()*,  
10424 *pthread\_condattr\_setclock()*, *pthread\_create()*, *pthread\_detach()*, *pthread\_equal()*, *pthread\_exit()*,  
10425 *pthread\_getconcurrency()*, *pthread\_getcpuclockid()*, *pthread\_getschedparam()*, *pthread\_join()*,  
10426 *pthread\_key\_create()*, *pthread\_key\_delete()*, *pthread\_mutex\_init()*, *pthread\_mutex\_lock()*,  
10427 *pthread\_mutex\_setprioceiling()*, *pthread\_mutex\_timedlock()*, *pthread\_mutexattr\_init()*,  
10428 *pthread\_mutexattr\_gettype()*, *pthread\_mutexattr\_setprotocol()*, *pthread\_once()*,  
10429 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
10430 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,

10431 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*,  
 10432 *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_getpshared()*, *pthread\_rwlockattr\_init()*,  
 10433 *pthread\_rwlockattr\_setpshared()*, *pthread\_self()*, *pthread\_setcancelstate()*, *pthread\_setspecific()*,  
 10434 *pthread\_spin\_destroy()*, *pthread\_spin\_init()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*,  
 10435 *pthread\_spin\_unlock()*

## 10436 CHANGE HISTORY

10437 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 10438 Issue 6

10439 The RTT margin markers are broken out into their POSIX options.

10440 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
 10441 *pthread\_cond\_wait()* function.

10442 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the  
 10443 *pthread\_setschedparam()* function so that its second argument is of type **int**.

10444 The *pthread\_getcpuclockid()* and *pthread\_mutex\_timedlock()* functions are added for alignment  
 10445 with IEEE Std 1003.1d-1999.

10446 The following functions are added for alignment with IEEE Std 1003.1j-2000:

10447 *pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*,  
 10448 *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_init()*,  
 10449 *pthread\_barrierattr\_setpshared()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_setclock()*,  
 10450 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_spin\_destroy()*,  
 10451 *pthread\_spin\_init()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, and *pthread\_spin\_unlock()*.

10452 PTHREAD\_RWLOCK\_INITIALIZER is deleted for alignment with IEEE Std 1003.1j-2000.

10453 Functions previously marked as part of the Read-Write Locks option are now moved to the  
 10454 Threads option.

10455 The **restrict** keyword is added to the prototypes for *pthread\_attr\_getguardsize()*,  
 10456 *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedparam()*, *pthread\_attr\_getschedpolicy()*,  
 10457 *pthread\_attr\_getscope()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*,  
 10458 *pthread\_attr\_setschedparam()*, *pthread\_barrier\_init()*, *pthread\_barrierattr\_getpshared()*,  
 10459 *pthread\_cond\_init()*, *pthread\_cond\_signal()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*,  
 10460 *pthread\_condattr\_getclock()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*,  
 10461 *pthread\_getschedparam()*, *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_init()*,  
 10462 *pthread\_mutex\_setprioceiling()*, *pthread\_mutexattr\_getprioceiling()*, *pthread\_mutexattr\_getprotocol()*,  
 10463 *pthread\_mutexattr\_getpshared()*, *pthread\_mutexattr\_gettype()*, *pthread\_rwlock\_init()*,  
 10464 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlockattr\_getpshared()*, and  
 10465 *pthread\_sigmask()*.

10466 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and  
 10467 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI  
 10468 extension.

10469 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for  
 10470 the *pthread\_kill()* and *pthread\_sigmask()* functions. These are required to be in the <signal.h>  
 10471 header. They are allowed here through the name space rules.

10472 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread\_setschedprio()* function.

10473 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors 1  
 10474 that were in contradiction with the System Interfaces volume of IEEE Std 1003.1-2001. 1

## 10475 NAME

10476       pwd.h — password structure

## 10477 SYNOPSIS

10478       #include <pwd.h>

## 10479 DESCRIPTION

10480       The <pwd.h> header shall provide a definition for **struct passwd**, which shall include at least the  
10481       following members:

10482	char	*pw_name	User's login name.
10483	uid_t	pw_uid	Numerical user ID.
10484	gid_t	pw_gid	Numerical group ID.
10485	char	*pw_dir	Initial working directory.
10486	char	*pw_shell	Program to use as shell.

10487       The **gid\_t** and **uid\_t** types shall be defined as described in <sys/types.h>.

10488       The following shall be declared as functions and may also be defined as macros. Function  
10489       prototypes shall be provided.

10490	struct passwd *getpwnam(const char *);		
10491	struct passwd *getpwuid(uid_t);		
10492 TSF	int	getpwnam_r(const char *, struct passwd *, char *,	
10493		size_t, struct passwd **);	
10494	int	getpwuid_r(uid_t, struct passwd *, char *,	
10495		size_t, struct passwd **);	
10496 XSI	void	endpwent(void);	
10497	struct passwd *	getpwent(void);	
10498	void	setpwent(void);	
10499			

## 10500 APPLICATION USAGE

10501       None.

## 10502 RATIONALE

10503       None.

## 10504 FUTURE DIRECTIONS

10505       None.

## 10506 SEE ALSO

10507       <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *endpwent()*, *getpwnam()*,  
10508       *getpwuid()*

## 10509 CHANGE HISTORY

10510       First released in Issue 1.

### 10511 Issue 5

10512       The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

### 10513 Issue 6

10514       The following new requirements on POSIX implementations derive from alignment with the  
10515       Single UNIX Specification:

- 10516       • The **gid\_t** and **uid\_t** types are mandated.
- 10517       • The *getpwnam\_r()* and *getpwuid\_r()* functions are marked as part of the Thread-Safe  
10518       Functions option.

10519 **NAME**

10520        regex.h — regular expression matching types

10521 **SYNOPSIS**

10522        #include &lt;regex.h&gt;

10523 **DESCRIPTION**10524        The <regex.h> header shall define the structures and symbolic constants used by the *regcomp()*,  
10525        *regexexec()*, *regerror()*, and *regfree()* functions.10526        The structure type **regex\_t** shall contain at least the following member:

10527        size\_t        re\_nsub        Number of parenthesized subexpressions.

10528        The type **size\_t** shall be defined as described in <sys/types.h>.10529        The type **regoff\_t** shall be defined as a signed integer type that can hold the largest value that  
10530        can be stored in either a type **off\_t** or type **ssize\_t**. The structure type **regmatch\_t** shall contain  
10531        at least the following members:10532        regoff\_t        rm\_so        Byte offset from start of string  
10533                                       to start of substring.10534        regoff\_t        rm\_eo        Byte offset from start of string of the  
10535                                       first character after the end of substring.10536        Values for the *cflags* parameter to the *regcomp()* function are as follows:

10537        REG\_EXTENDED    Use Extended Regular Expressions.

10538        REG\_ICASE        Ignore case in match.

10539        REG\_NOSUB        Report only success or fail in *regexexec()*.

10540        REG\_NEWLINE     Change the handling of &lt;newline&gt;.

10541        Values for the *eflags* parameter to the *regexexec()* function are as follows:10542        REG\_NOTBOL       The circumflex character ('^'), when taken as a special character, does  
10543                                       not match the beginning of *string*.10544        REG\_NOTEOL       The dollar sign ('\$'), when taken as a special character, does not match  
10545                                       the end of *string*.

10546        The following constants shall be defined as error return values:

10547        REG\_NOMATCH     *regexexec()* failed to match.

10548        REG\_BADPAT       Invalid regular expression.

10549        REG\_ECOLLATE     Invalid collating element referenced.

10550        REG\_ETYPE        Invalid character class type referenced.

10551        REG\_EESCAPE       Trailing '\\' in pattern.

10552        REG\_ESUBREG       Number in *\digit* invalid or in error.

10553        REG\_EBRACK       "[ ]" imbalance.

10554        REG\_EPAREN       "\ ( \ )" or "\ ( )" imbalance.

10555        REG\_EBRACE       "\ { \ }" imbalance.

10556        REG\_BADBR        Content of "\ { \ }" invalid: not a number, number too large, more than  
10557                                       two numbers, first larger than second.

10558	REG_ERANGE	Invalid endpoint in range expression.
10559	REG_ESPACE	Out of memory.
10560	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
10561	OB REG_ENOSYS	Reserved.
10562	The following shall be declared as functions and may also be defined as macros. Function	
10563	prototypes shall be provided.	
10564	int regcomp(regex_t *restrict, const char *restrict, int);	
10565	size_t regerror(int, const regex_t *restrict, char *restrict, size_t);	
10566	int regexec(const regex_t *restrict, const char *restrict, size_t,	
10567	regmatch_t[restrict], int);	
10568	void regfree(regex_t *);	
10569	The implementation may define additional macros or constants using names beginning with	
10570	REG_.	
10571	<b>APPLICATION USAGE</b>	
10572	None.	
10573	<b>RATIONALE</b>	
10574	None.	
10575	<b>FUTURE DIRECTIONS</b>	
10576	None.	
10577	<b>SEE ALSO</b>	
10578	<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, <i>regcomp()</i> , the Shell and	
10579	Utilities volume of IEEE Std 1003.1-2001	
10580	<b>CHANGE HISTORY</b>	
10581	First released in Issue 4.	
10582	Originally derived from the ISO POSIX-2 standard.	
10583	<b>Issue 6</b>	
10584	The REG_ENOSYS constant is marked obsolescent.	
10585	The <b>restrict</b> keyword is added to the prototypes for <i>regcomp()</i> , <i>regerror()</i> , and <i>regexec()</i> .	
10586	A statement is added that the <b>size_t</b> type is defined as described in <sys/types.h>.	

## 10587 NAME

10588 sched.h — execution scheduling (**REALTIME**)

## 10589 SYNOPSIS

10590 PS #include &lt;sched.h&gt;

10591

## 10592 DESCRIPTION

10593 The <sched.h> header shall define the **sched\_param** structure, which contains the scheduling  
 10594 parameters required for implementation of each supported scheduling policy. This structure  
 10595 shall contain at least the following member:

10596 THR int sched\_priority Process or thread execution scheduling priority. 2

10597 SS|TSP In addition, if **\_POSIX\_SPORADIC\_SERVER** or **\_POSIX\_THREAD\_SPORADIC\_SERVER** is  
 10598 defined, the **sched\_param** structure defined in <sched.h> shall contain the following members  
 10599 in addition to those specified above:

10600	int	sched_ss_low_priority	Low scheduling priority for
10601			sporadic server.
10602	struct timespec	sched_ss_repl_period	Replenishment period for
10603			sporadic server.
10604	struct timespec	sched_ss_init_budget	Initial budget for sporadic server.
10605	int	sched_ss_max_repl	Maximum pending replenishments for
10606			sporadic server.

10607

10608 THR Each process or thread is controlled by an associated scheduling policy and priority. Associated 2  
 10609 with each policy is a priority range. Each policy definition specifies the minimum priority range  
 10610 for that policy. The priority ranges for each policy may overlap the priority ranges of other  
 10611 policies.

10612 Four scheduling policies are defined; others may be defined by the implementation. The four  
 10613 standard policies are indicated by the values of the following symbolic constants:

10614 SCHED\_FIFO First in-first out (FIFO) scheduling policy.

10615 SCHED\_RR Round robin scheduling policy.

10616 SS|TSP SCHED\_SPORADIC Sporadic server scheduling policy.

10617 SCHED\_OTHER Another scheduling policy.

10618 The values of these constants are distinct.

10619 The following shall be declared as functions and may also be defined as macros. Function  
 10620 prototypes shall be provided.

10621	TPS	int	sched_get_priority_max(int);	1
10622		int	sched_get_priority_min(int);	
10623		int	sched_getparam(pid_t, struct sched_param *);	1
10624		int	sched_getscheduler(pid_t);	
10625	TPS	int	sched_rr_get_interval(pid_t, struct timespec *);	1
10626		int	sched_setparam(pid_t, const struct sched_param *);	1
10627		int	sched_setscheduler(pid_t, int, const struct sched_param *);	
10628	THR	int	sched_yield(void);	1
10629				



10630	Inclusion of the <sched.h> header may make visible all symbols from the <time.h> header.	
10631	<b>APPLICATION USAGE</b>	
10632	None.	
10633	<b>RATIONALE</b>	
10634	None.	
10635	<b>FUTURE DIRECTIONS</b>	
10636	None.	
10637	<b>SEE ALSO</b>	
10638	<time.h>	
10639	<b>CHANGE HISTORY</b>	
10640	First released in Issue 5. Included for alignment with the POSIX Realtime Extension.	
10641	<b>Issue 6</b>	
10642	The <sched.h> header is marked as part of the Process Scheduling option.	
10643	Sporadic server members are added to the <b>sched_param</b> structure, and the SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.	
10644		
10645	IEEE PASC Interpretation 1003.1 #108 is applied, correcting the <b>sched_param</b> structure whose members <i>sched_ss_repl_period</i> and <i>sched_ss_init_budget</i> should be type <b>struct timespec</b> and not <b>timespec</b> .	
10646		
10647		
10648	Symbols from <time.h> may be made visible when <sched.h> is included.	
10649	IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied,	1
10650	aligning the function prototype shading and margin codes with the System Interfaces volume of	1
10651	IEEE Std 1003.1-2001.	1
10652	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/23 is applied, updating the	2
10653	DESCRIPTION to differentiate between thread and process execution.	2

10654 **NAME**

10655 search.h — search tables

10656 **SYNOPSIS**10657 XSI `#include <search.h>`

10658

10659 **DESCRIPTION**

10660 The <search.h> header shall define the **ENTRY** type for structure **entry** which shall include the  
 10661 following members:

```
10662     char    *key
10663     void    *data
```

10664 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as  
 10665 follows:

```
10666     enum { FIND, ENTER } ACTION;
10667     enum { preorder, postorder, endorder, leaf } VISIT;
```

10668 The **size\_t** type shall be defined as described in <sys/types.h>.

10669 The following shall be declared as functions and may also be defined as macros. Function  
 10670 prototypes shall be provided.

```
10671     int      hcreate(size_t);
10672     void     hdestroy(void);
10673     ENTRY *hsearch(ENTRY, ACTION);
10674     void     insque(void *, void *);
10675     void     *lfind(const void *, const void *, size_t *,
10676                   size_t, int (*)(const void *, const void *));
10677     void     *lsearch(const void *, void *, size_t *,
10678                   size_t, int (*)(const void *, const void *));
10679     void     remque(void *);
10680     void     *tdelete(const void *restrict, void **restrict,
10681                   int (*)(const void *, const void *));
10682     void     *tfind(const void *, void *const *,
10683                   int (*)(const void *, const void *));
10684     void     *tsearch(const void *, void **,
10685                   int (*)(const void *, const void *));
10686     void     twalk(const void *,
10687                   void (*)(const void *, VISIT, int ));
```

10688 **APPLICATION USAGE**

10689 None.

10690 **RATIONALE**

10691 None.

10692 **FUTURE DIRECTIONS**

10693 None.

10694 **SEE ALSO**

10695 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *hcreate()*, *insque()*,  
 10696 *lsearch()*, *remque()*, *tsearch()*

10697 **CHANGE HISTORY**

10698 First released in Issue 1. Derived from Issue 1 of the SVID.

10699 **Issue 6**

10700 The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete()* and  
10701 *tsearch()*.

10702 The **restrict** keyword is added to the prototype for *tdelete()*.

## 10703 NAME

10704 semaphore.h — semaphores (REALTIME)

## 10705 SYNOPSIS

10706 SEM `#include <semaphore.h>`

10707

## 10708 DESCRIPTION

10709 The <semaphore.h> header shall define the **sem\_t** type, used in performing semaphore  
 10710 operations. The semaphore may be implemented using a file descriptor, in which case  
 10711 applications are able to open up at least a total of {OPEN\_MAX} files and semaphores. The  
 10712 symbol SEM\_FAILED shall be defined (see *sem\_open()*).

10713 The following shall be declared as functions and may also be defined as macros. Function  
 10714 prototypes shall be provided.

```
10715 int      sem_close(sem_t *);
10716 int      sem_destroy(sem_t *);
10717 int      sem_getvalue(sem_t *restrict, int *restrict);
10718 int      sem_init(sem_t *, int, unsigned);
10719 sem_t *sem_open(const char *, int, ...);
10720 int      sem_post(sem_t *);
10721 TMO int    sem_timedwait(sem_t *restrict, const struct timespec *restrict);
10722 int      sem_trywait(sem_t *);
10723 int      sem_unlink(const char *);
10724 int      sem_wait(sem_t *);
```

10725 Inclusion of the <semaphore.h> header may make visible symbols defined in the headers  
 10726 <fcntl.h> and <sys/types.h>.

## 10727 APPLICATION USAGE

10728 None.

## 10729 RATIONALE

10730 None.

## 10731 FUTURE DIRECTIONS

10732 None.

## 10733 SEE ALSO

10734 <fcntl.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *sem\_destroy()*,  
 10735 *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*,  
 10736 *sem\_wait()*

## 10737 CHANGE HISTORY

10738 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

## 10739 Issue 6

10740 The &lt;semaphore.h&gt; header is marked as part of the Semaphores option.

10741 The Open Group Corrigendum U021/3 is applied, adding a description of SEM\_FAILED.

10742 The *sem\_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.10743 The **restrict** keyword is added to the prototypes for *sem\_getvalue()* and *sem\_timedwait()*.

10744 **NAME**

10745       setjmp.h — stack environment declarations

10746 **SYNOPSIS**

10747       #include <setjmp.h>

10748 **DESCRIPTION**

10749 CX       Some of the functionality described on this reference page extends the ISO C standard.  
10750       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
10751       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
10752       symbols in this header.

10753 CX       The <setjmp.h> header shall define the array types **jmp\_buf** and **sigjmp\_buf**.

10754       The following shall be declared as functions and may also be defined as macros. Function  
10755       prototypes shall be provided.

10756       void    longjmp(jmp\_buf, int);  
10757 CX       void   siglongjmp(sigjmp\_buf, int);  
10758 XSI       void   \_longjmp(jmp\_buf, int);  
10759

10760       The following may be declared as a function, or defined as a macro, or both. Function prototypes  
10761       shall be provided.

10762       int     setjmp(jmp\_buf);  
10763 CX       int   sigsetjmp(sigjmp\_buf, int);  
10764 XSI       int   \_setjmp(jmp\_buf);  
10765

10766 **APPLICATION USAGE**

10767       None.

10768 **RATIONALE**

10769       None.

10770 **FUTURE DIRECTIONS**

10771       None.

10772 **SEE ALSO**

10773       The System Interfaces volume of IEEE Std 1003.1-2001, *longjmp()*, *\_longjmp()*, *setjmp()*,  
10774       *siglongjmp()*, *sigsetjmp()*

10775 **CHANGE HISTORY**

10776       First released in Issue 1.

10777 **Issue 6**

10778       Extensions beyond the ISO C standard are marked.

## 10779 NAME

10780 signal.h — signals

## 10781 SYNOPSIS

10782 #include &lt;signal.h&gt;

## 10783 DESCRIPTION

10784 CX Some of the functionality described on this reference page extends the ISO C standard.  
 10785 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 10786 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
 10787 symbols in this header.

10788 The <signal.h> header shall define the following symbolic constants, each of which expands to a  
 10789 distinct constant expression of the type:

10790 void (\*)(int)

10791 whose value matches no declarable function.

10792 SIG\_DFL Request for default signal handling.

10793 SIG\_ERR Return value from *signal()* in case of error.

10794 CX SIG\_HOLD Request that signal be held.

10795 SIG\_IGN Request that signal be ignored.

10796 The following data types shall be defined through **typedef**:

10797 **sig\_atomic\_t** Possibly volatile-qualified integer type of an object that can be accessed as  
 10798 an atomic entity, even in the presence of asynchronous interrupts.

10799 CX **sigset\_t** Integer or structure type of an object used to represent sets of signals.10800 CX **pid\_t** As described in <sys/types.h>.

10801 RTS The <signal.h> header shall define the **sigevent** structure, which has at least the following  
 10802 members:

10803	int	sigev_notify	Notification type.
10804	int	sigev_signo	Signal number.
10805	union sigval	sigev_value	Signal value.
10806	void (*)(union sigval)	sigev_notify_function	Notification function.
10807	(pthread_attr_t *)	sigev_notify_attributes	Notification attributes.

10808 The following values of *sigev\_notify* shall be defined:

10809 SIGEV\_NONE No asynchronous notification is delivered when the event of interest  
 10810 occurs.

10811 SIGEV\_SIGNAL A queued signal, with an application-defined value, is generated when  
 10812 the event of interest occurs.

10813 SIGEV\_THREAD A notification function is called to perform notification.

10814 The **sigval** union shall be defined as:

10815	int	sival_int	Integer signal value.
10816	void *	sival_ptr	Pointer signal value.

10817 This header shall also declare the macros SIGRTMIN and SIGRTMAX, which evaluate to integer  
 10818 expressions, and specify a range of signal numbers that are reserved for application use and for  
 10819 which the realtime signal behavior specified in this volume of IEEE Std 1003.1-2001 is supported.

10820 The signal numbers in this range do not overlap any of the signals specified in the following  
10821 table.

10822 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG\_MAX} signal  
10823 numbers.

10824 It is implementation-defined whether realtime signal behavior is supported for other signals.

10825 This header also declares the constants that are used to refer to the signals that occur in the  
10826 system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive  
10827 integer values. The value 0 is reserved for use as the null signal (see *kill()*). Additional  
10828 implementation-defined signals may occur in the system.

10829 CX The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT,  
10830 SIGSEGV, and SIGTERM to be defined.

10831 The following signals shall be supported on all implementations (default actions are explained  
10832 below the table):

10833

Signal	Default Action	Description
SIGABRT	A	Process abort signal.
SIGALRM	T	Alarm clock.
SIGBUS	A	Access to an undefined portion of a memory object.
SIGCHLD	I	Child process terminated, stopped, or continued.
SIGCONT	C	Continue executing, if stopped.
SIGFPE	A	Erroneous arithmetic operation.
SIGHUP	T	Hangup.
SIGILL	A	Illegal instruction.
SIGINT	T	Terminal interrupt signal.
SIGKILL	T	Kill (cannot be caught or ignored).
SIGPIPE	T	Write on a pipe with no one to read it.
SIGQUIT	A	Terminal quit signal.
SIGSEGV	A	Invalid memory reference.
SIGSTOP	S	Stop executing (cannot be caught or ignored).
SIGTERM	T	Termination signal.
SIGTSTP	S	Terminal stop signal.
SIGTTIN	S	Background process attempting read.
SIGTTOU	S	Background process attempting write.
SIGUSR1	T	User-defined signal 1.
SIGUSR2	T	User-defined signal 2.
SIGPOLL	T	Pollable event.
SIGPROF	T	Profiling timer expired.
SIGSYS	A	Bad system call.
SIGTRAP	A	Trace/breakpoint trap.
SIGURG	I	High bandwidth data is available at a socket.
SIGVTALRM	T	Virtual timer expired.
SIGXCPU	A	CPU time limit exceeded.
SIGXFSZ	A	File size limit exceeded.

10864 The default actions are as follows:

10865 T Abnormal termination of the process. The process is terminated with all the consequences  
10866 of *\_exit()* except that the status made available to *wait()* and *waitpid()* indicates abnormal  
10867 termination by the specified signal.

10868	A	Abnormal termination of the process.	
10869 XSI		Additionally, implementation-defined abnormal termination actions, such as creation of a	
10870		<b>core</b> file, may occur.	
10871	I	Ignore the signal.	
10872	S	Stop the process.	
10873	C	Continue the process, if it is stopped; otherwise, ignore the signal.	
10874 CX		The header shall provide a declaration of <b>struct sigaction</b> , including at least the following	
10875		members:	
10876		void (*sa_handler)(int)	Pointer to a signal-catching function or one of the macros
10877			SIG_IGN or SIG_DFL.
10878		sigset_t sa_mask	Set of signals to be blocked during execution of the signal
10879			handling function.
10880		int sa_flags	Special flags.
10881		void (*sa_sigaction)(int, siginfo_t *, void *)	
10882			Pointer to a signal-catching function.
10883			
10884 XSI		The storage occupied by <i>sa_handler</i> and <i>sa_sigaction</i> may overlap, and a conforming application	
10885		shall not use both simultaneously.	
10886		The following shall be declared as constants:	
10887 CX	SA_NOCLDSTOP	Do not generate SIGCHLD when children stop	
10888 XSI		or stopped children continue.	
10889 CX	SIG_BLOCK	The resulting set is the union of the current set and the signal set pointed	
10890		to by the argument <i>set</i> .	
10891 CX	SIG_UNBLOCK	The resulting set is the intersection of the current set and the complement	
10892		of the signal set pointed to by the argument <i>set</i> .	
10893 CX	SIG_SETMASK	The resulting set is the signal set pointed to by the argument <i>set</i> .	
10894 XSI	SA_ONSTACK	Causes signal delivery to occur on an alternate stack.	
10895 XSI	SA_RESETHAND	Causes signal dispositions to be set to SIG_DFL on entry to signal	
10896		handlers.	
10897 XSI	SA_RESTART	Causes certain functions to become restartable.	
10898 XSI	SA_SIGINFO	Causes extra information to be passed to signal handlers at the time of	
10899		receipt of a signal.	
10900 XSI	SA_NOCLDWAIT	Causes implementations not to create zombie processes on child death.	
10901 XSI	SA_NODEFER	Causes signal not to be automatically blocked on entry to signal handler.	
10902 XSI	SS_ONSTACK	Process is executing on an alternate signal stack.	
10903 XSI	SS_DISABLE	Alternate signal stack is disabled.	
10904 XSI	MINSIGSTKSZ	Minimum stack size for a signal handler.	
10905 XSI	SIGSTKSZ	Default size in bytes for the alternate signal stack.	
10906 XSI		The <b>ucontext_t</b> structure shall be defined through <b>typedef</b> as described in <ucontext.h>.	
10907		The <b>mcontext_t</b> type shall be defined through <b>typedef</b> as described in <ucontext.h>.	



10908 The <signal.h> header shall define the **stack\_t** type as a structure that includes at least the  
 10909 following members:

10910	void	*ss_sp	Stack base or pointer.
10911	size_t	ss_size	Stack size.
10912	int	ss_flags	Flags.

10913 The <signal.h> header shall define the **sigstack** structure that includes at least the following  
 10914 members:

10915	int	ss_onstack	Non-zero when signal stack is in use.
10916	void	*ss_sp	Signal stack pointer.

10917

10918 CX The <signal.h> header shall define the **siginfo\_t** type as a structure that includes at least the  
 10919 following members:

10920 CX	int	si_signo	Signal number.	
10921	int	si_code	Signal code.	2
10922 XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with this signal, as defined in <errno.h>.	2
10923				
10924	pid_t	si_pid	Sending process ID.	2
10925	uid_t	si_uid	Real user ID of sending process.	
10926	void	*si_addr	Address of faulting instruction.	
10927	int	si_status	Exit value or signal.	
10928	long	si_band	Band event for SIGPOLL.	
10929 RTS	union sigval	si_value	Signal value.	
10930				

10931 The macros specified in the **Code** column of the following table are defined for use as values of  
 10932 XSI *si\_code* that are signal-specific or non-signal-specific reasons why the signal was generated.

10933

10934

10935 XSI

10936

10937

10938

10939

10940

10941

10942

10943

10944

10945

10946

10947

10948

10949

10950

10951

10952

10953

10954

10955

10956

10957

10958

10959

10960

10961

10962

10963

10964

10965

10966

10967

10968

10969

10970 CX

10971

10972

10973

10974

10975

10976

Signal	Code	Reason
SIGILL	ILL_ILLOPC	Illegal opcode.
	ILL_ILLOPN	Illegal operand.
	ILL_ILLADR	Illegal addressing mode.
	ILL_ILLTRP	Illegal trap.
	ILL_PRVOPC	Privileged opcode.
	ILL_PRVREG	Privileged register.
	ILL_COPROC	Coprocessor error.
	ILL_BADSTK	Internal stack error.
SIGFPE	FPE_INTDIV	Integer divide by zero.
	FPE_INTOVF	Integer overflow.
	FPE_FLTDIV	Floating-point divide by zero.
	FPE_FLTOVF	Floating-point overflow.
	FPE_FLTUND	Floating-point underflow.
	FPE_FLTRES	Floating-point inexact result.
	FPE_FLTINV	Invalid floating-point operation.
	FPE_FLTSUB	Subscript out of range.
SIGSEGV	SEGV_MAPERR	Address not mapped to object.
	SEGV_ACCERR	Invalid permissions for mapped object.
SIGBUS	BUS_ADRALN	Invalid address alignment.
	BUS_ADRERR	Nonexistent physical address.
	BUS_OBJERR	Object-specific hardware error.
SIGTRAP	TRAP_BRKPT	Process breakpoint.
	TRAP_TRACE	Process trace trap.
SIGCHLD	CLD_EXITED	Child has exited.
	CLD_KILLED	Child has terminated abnormally and did not create a <b>core</b> file.
	CLD_DUMPED	Child has terminated abnormally and created a <b>core</b> file.
	CLD_TRAPPED	Traced child has trapped.
	CLD_STOPPED	Child has stopped.
	CLD_CONTINUED	Stopped child has continued.
SIGPOLL	POLL_IN	Data input available.
	POLL_OUT	Output buffers available.
	POLL_MSG	Input message available.
	POLL_ERR	I/O error.
	POLL_PRI	High priority input available.
	POLL_HUP	Device disconnected.
Any	SI_USER	Signal sent by <i>kill()</i> .
	SI_QUEUE	Signal sent by the <i>sigqueue()</i> .
	SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
	SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
	SI_MESGQ	Signal generated by arrival of a message on an empty message queue.

10977 XSI

10978

10979

10980

10981

Implementations may support additional *si\_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

10982 In addition, the following signal-specific information shall be available:

Signal	Member	Value
SIGILL SIGFPE	<b>void * <i>si_addr</i></b>	Address of faulting instruction.
SIGSEGV SIGBUS	<b>void * <i>si_addr</i></b>	Address of faulting memory reference.
SIGCHLD	<b>pid_t <i>si_pid</i></b> <b>int <i>si_status</i></b> <b>uid_t <i>si_uid</i></b>	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
SIGPOLL	<b>long <i>si_band</i></b>	Band event for POLL_IN, POLL_OUT, or POLL_MSG.

10993 For some implementations, the value of *si\_addr* may be inaccurate.

10994 The following shall be declared as functions and may also be defined as macros:

```

10995 XSI void (*bsd_signal(int, void (*)(int)))(int);
10996 CX   int kill(pid_t, int);
10997 XSI   int killpg(pid_t, int);
10998 THR   int pthread_kill(pthread_t, int);
10999     int pthread_sigmask(int, const sigset_t *, sigset_t *);
11000     int raise(int);
11001 CX     int sigaction(int, const struct sigaction *restrict,
11002                     struct sigaction *restrict);
11003     int sigaddset(sigset_t *, int);
11004 XSI     int sigaltstack(const stack_t *restrict, stack_t *restrict);
11005 CX     int sigdelset(sigset_t *, int);
11006     int sigemptyset(sigset_t *);
11007     int sigfillset(sigset_t *);
11008 XSI     int sighold(int);
11009     int sigignore(int);
11010     int siginterrupt(int, int);
11011 CX     int sigismember(const sigset_t *, int);
11012     void (*signal(int, void (*)(int)))(int);
11013 XSI     int sigpause(int);
11014 CX     int sigpending(sigset_t *);
11015     int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
11016 RTS     int sigqueue(pid_t, int, const union sigval);
11017 XSI     int sigrelse(int);
11018     void (*sigset(int, void (*)(int)))(int);
11019 CX     int sigsuspend(const sigset_t *);
11020 RTS     int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
11021                         const struct timespec *restrict);
11022 CX     int sigwait(const sigset_t *restrict, int *restrict);
11023 RTS     int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
11024

```

11025 CX Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.

## 11026 APPLICATION USAGE

11027 None.

## 11028 RATIONALE

11029 None.

## 11030 FUTURE DIRECTIONS

11031 None.

## 11032 SEE ALSO

11033 <errno.h>, <stropts.h>, <sys/types.h>, <time.h>, <ucontext.h>, the System Interfaces volume of  
 11034 IEEE Std 1003.1-2001, *alarm()*, *bsd\_signal()*, *ioctl()*, *kill()*, *killpg()*, *raise()*, *sigaction()*, *sigaddset()*,  
 11035 *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *siginterrupt()*, *sigismember()*, *signal()*,  
 11036 *sigpending()*, *sigprocmask()*, *sigqueue()*, *sigsuspend()*, *sigwaitinfo()*, *wait()*, *waitid()*

## 11037 CHANGE HISTORY

11038 First released in Issue 1.

## 11039 Issue 5

11040 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 11041 Threads Extension.

11042 The default action for SIGURG is changed from i to iii. The function prototype for *sigmask()* is  
 11043 removed.

## 11044 Issue 6

11045 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for  
 11046 abnormal termination is clarified.

11047 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()*  
 11048 function.

11049 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev\_notify\_function*  
 11050 function member of the **sigevent** structure.

11051 The following new requirements on POSIX implementations derive from alignment with the  
 11052 Single UNIX Specification:

11053 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now  
 11054 mandated. This is also a FIPS requirement.

11055 • The **pid\_t** definition is mandated.

11056 The RT markings are changed to RTS to denote that the semantics are part of the Realtime 1  
 11057 Signals Extension option.

11058 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,  
 11059 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

11060 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from  
 11061 <time.h> may be made visible when <signal.h> is included.

11062 Extensions beyond the ISO C standard are marked.

11063 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive 1  
 11064 text for members of **struct sigaction**. 1

11065 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of 1  
 11066 the *sa\_sigaction* member of **struct sigaction**. 1

11067 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/24 is applied, reworking the ordering of 2  
 11068 the **siginfo\_t** type structure in the DESCRIPTION. This is an editorial change and no normative 2

11069

change is intended.

2

## 11070 NAME

11071 spawn.h — spawn (ADVANCED REALTIME)

## 11072 SYNOPSIS

11073 SPN #include &lt;spawn.h&gt;

11074

## 11075 DESCRIPTION

11076 The <spawn.h> header shall define the **posix\_spawnattr\_t** and **posix\_spawn\_file\_actions\_t**  
 11077 types used in performing spawn operations.

11078 The <spawn.h> header shall define the flags that may be set in a **posix\_spawnattr\_t** object using  
 11079 the *posix\_spawnattr\_setflags()* function:

11080 POSIX\_SPAWN\_RESETEIDS

11081 POSIX\_SPAWN\_SETPGROUP

11082 PS POSIX\_SPAWN\_SETSCHEDPARAM

11083 POSIX\_SPAWN\_SETSCHEDULER

11084 POSIX\_SPAWN\_SETSIGDEF

11085 POSIX\_SPAWN\_SETSIGMASK

11086 The following shall be declared as functions and may also be defined as macros. Function  
 11087 prototypes shall be provided.

```

11088 int    posix_spawn(pid_t *restrict, const char *restrict,
11089                  const posix_spawn_file_actions_t *,
11090                  const posix_spawnattr_t *restrict, char *const [restrict],
11091                  char *const [restrict]);
11092 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11093                  int);
11094 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11095                  int, int);
11096 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11097                  int, const char *restrict, int, mode_t);
11098 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11099 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11100 int    posix_spawnattr_destroy(posix_spawnattr_t *);
11101 int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11102                  sigset_t *restrict);
11103 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11104                  short *restrict);
11105 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11106                  pid_t *restrict);
11107 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11108                  struct sched_param *restrict);
11109 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11110                  int *restrict);
11111 int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11112                  sigset_t *restrict);
11113 int    posix_spawnattr_init(posix_spawnattr_t *);
11114 int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11115                  const sigset_t *restrict);
11116 int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11117 int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);

```

```

11118 PS      int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11119           const struct sched_param *restrict);
11120      int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11121      int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11122           const sigset_t *restrict);
11123      int    posix_spawn(pid_t *restrict, const char *restrict,
11124           const posix_spawn_file_actions_t *,
11125           const posix_spawnattr_t *restrict,
11126           char *const [restrict], char *const [restrict]);

```

11127 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h>,  
11128 <signal.h>, and <sys/types.h> headers.

## 11129 APPLICATION USAGE

11130 None.

## 11131 RATIONALE

11132 None.

## 11133 FUTURE DIRECTIONS

11134 None.

## 11135 SEE ALSO

11136 <sched.h>, <semaphore.h>, <signal.h>, <sys/types.h>, the System Interfaces volume of  
11137 IEEE Std 1003.1-2001, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
11138 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
11139 *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_init()*,  
11140 *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*, *posix\_spawnattr\_setpgroup()*,  
11141 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
11142 *posix\_spawn()*, *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_adddup2()*,  
11143 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawn\_file\_actions\_destroy()*,  
11144 *posix\_spawn\_file\_actions\_init()*, *posix\_spawnnp()*

## 11145 CHANGE HISTORY

11146 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11147 The **restrict** keyword is added to the prototypes for *posix\_spawn()*,  
11148 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
11149 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
11150 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setschedparam()*,  
11151 *posix\_spawnattr\_setsigmask()*, and *posix\_spawnnp()*.

## 11152 NAME

11153        `stdarg.h` — handle variable argument list

## 11154 SYNOPSIS

11155        `#include <stdarg.h>`11156        `void va_start(va_list ap, argN);`11157        `void va_copy(va_list dest, va_list src);`11158        `type va_arg(va_list ap, type);`11159        `void va_end(va_list ap);`

## 11160 DESCRIPTION

11161 CX     The functionality described on this reference page is aligned with the ISO C standard. Any  
 11162     conflict between the requirements described here and the ISO C standard is unintentional. This  
 11163     volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11164     The `<stdarg.h>` header shall contain a set of macros which allows portable functions that accept  
 11165     variable argument lists to be written. Functions that have variable argument lists (such as  
 11166     `printf()`) but do not use these macros are inherently non-portable, as different systems use  
 11167     different argument-passing conventions.

11168     The type `va_list` shall be defined for variables used to traverse the list.

11169     The `va_start()` macro is invoked to initialize `ap` to the beginning of the list before any calls to  
 11170     `va_arg()`.

11171     The `va_copy()` macro initializes `dest` as a copy of `src`, as if the `va_start()` macro had been applied  
 11172     to `dest` followed by the same sequence of uses of the `va_arg()` macro as had previously been used  
 11173     to reach the present state of `src`. Neither the `va_copy()` nor `va_start()` macro shall be invoked to  
 11174     reinitialize `dest` without an intervening invocation of the `va_end()` macro for the same `dest`.

11175     The object `ap` may be passed as an argument to another function; if that function invokes the  
 11176     `va_arg()` macro with parameter `ap`, the value of `ap` in the calling function is unspecified and shall  
 11177     be passed to the `va_end()` macro prior to any further reference to `ap`. The parameter `argN` is the  
 11178     identifier of the rightmost parameter in the variable parameter list in the function definition (the  
 11179     one just before the `...`). If the parameter `argN` is declared with the **register** storage class, with a  
 11180     function type or array type, or with a type that is not compatible with the type that results after  
 11181     application of the default argument promotions, the behavior is undefined.

11182     The `va_arg()` macro shall return the next argument in the list pointed to by `ap`. Each invocation  
 11183     of `va_arg()` modifies `ap` so that the values of successive arguments are returned in turn. The `type`  
 11184     parameter shall be a type name specified such that the type of a pointer to an object that has the  
 11185     specified type can be obtained simply by postfixing a `'*'` to `type`. If there is no actual next  
 11186     argument, or if `type` is not compatible with the type of the actual next argument (as promoted  
 11187     according to the default argument promotions), the behavior is undefined, except for the  
 11188     following cases:

- 11189        • One type is a signed integer type, the other type is the corresponding unsigned integer type,  
 11190        and the value is representable in both types.
- 11191        • One type is a pointer to **void** and the other is a pointer to a character type.
- 11192 XSI    • Both types are pointers.

11193     Different types can be mixed, but it is up to the routine to know what type of argument is  
 11194     expected.

11195     The `va_end()` macro is used to clean up; it invalidates `ap` for use (unless `va_start()` or `va_copy()` is  
 11196     invoked again).



11197 Each invocation of the *va\_start()* and *va\_copy()* macros shall be matched by a corresponding  
11198 invocation of the *va\_end()* macro in the same function.

11199 Multiple traversals, each bracketed by *va\_start()* ... *va\_end()*, are possible.

#### 11200 EXAMPLES

11201 This example is a possible implementation of *execl()*:

```
11202 #include <stdarg.h>
11203 #define MAXARGS 31
11204 /*
11205  * execl is called by
11206  * execl(file, arg1, arg2, ..., (char *) (0));
11207  */
11208 int execl(const char *file, const char *args, ...)
11209 {
11210     va_list ap;
11211     char *array[MAXARGS + 1];
11212     int argno = 0;
11213
11214     va_start(ap, args);
11215     while (args != 0 && argno < MAXARGS)
11216     {
11217         array[argno++] = args;
11218         args = va_arg(ap, const char *);
11219     }
11220     array[argno] = (char *) 0;
11221     va_end(ap);
11222     return execv(file, array);
11223 }
```

#### 11223 APPLICATION USAGE

11224 It is up to the calling routine to communicate to the called routine how many arguments there  
11225 are, since it is not always possible for the called routine to determine this in any other way. For  
11226 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell  
11227 how many arguments are there by the *format* argument.

#### 11228 RATIONALE

11229 None.

#### 11230 FUTURE DIRECTIONS

11231 None.

#### 11232 SEE ALSO

11233 The System Interfaces volume of IEEE Std 1003.1-2001, *exec*, *printf()*

#### 11234 CHANGE HISTORY

11235 First released in Issue 4. Derived from the ANSI C standard.

#### 11236 Issue 6

11237 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

11238 **NAME**

11239       stdbool.h — boolean type and values

11240 **SYNOPSIS**

11241       #include &lt;stdbool.h&gt;

11242 **DESCRIPTION**

11243 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
11244       conflict between the requirements described here and the ISO C standard is unintentional. This  
11245       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11246       The &lt;stdbool.h&gt; header shall define the following macros:

11247       **bool**       Expands to **\_Bool**.11248       **true**       Expands to the integer constant 1.11249       **false**       Expands to the integer constant 0.11250       **\_\_bool\_true\_false\_are\_defined**

11251               Expands to the integer constant 1.

11252       An application may undefine and then possibly redefine the macros **bool**, **true**, and **false**.11253 **APPLICATION USAGE**

11254       None.

11255 **RATIONALE**

11256       None.

11257 **FUTURE DIRECTIONS**

11258       The ability to undefine and redefine the macros **bool**, **true**, and **false** is an obsolescent feature  
11259       and may be withdrawn in a future version.

11260 **SEE ALSO**

11261       None.

11262 **CHANGE HISTORY**

11263       First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11264 **NAME**

11265         **stddef.h** — standard type definitions

11266 **SYNOPSIS**

11267         #include <stddef.h>

11268 **DESCRIPTION**

11269 **CX**         The functionality described on this reference page is aligned with the ISO C standard. Any  
11270 conflict between the requirements described here and the ISO C standard is unintentional. This  
11271 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11272         The <stddef.h> header shall define the following macros:

11273         **NULL**         Null pointer constant.

11274         **offsetof**(*type*, *member-designator*)

11275                 Integer constant expression of type **size\_t**, the value of which is the offset in bytes  
11276 to the structure member (*member-designator*), from the beginning of its structure  
11277 (*type*).

11278         The <stddef.h> header shall define the following types:

11279         **ptrdiff\_t**     Signed integer type of the result of subtracting two pointers.

11280         **wchar\_t**     Integer type whose range of values can represent distinct wide-character codes for  
11281 all members of the largest character set specified among the locales supported by  
11282 the compilation environment: the null character has the code value 0 and each  
11283 member of the portable character set has a code value equal to its value when used  
11284 as the lone character in an integer character constant.

11285         **size\_t**        Unsigned integer type of the result of the *sizeof* operator.

11286         The implementation shall support one or more programming environments in which the widths  
11287 of **ptrdiff\_t**, **size\_t**, and **wchar\_t** are no greater than the width of type **long**. The names of these  
11288 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

11289 **APPLICATION USAGE**

11290         None.

11291 **RATIONALE**

11292         None.

11293 **FUTURE DIRECTIONS**

11294         None.

11295 **SEE ALSO**

11296         <**wchar.h**>, <**sys/types.h**>, the System Interfaces volume of IEEE Std 1003.1-2001, *confstr()*, the  
11297 Shell and Utilities volume of IEEE Std 1003.1-2001, *getconf*

11298 **CHANGE HISTORY**

11299         First released in Issue 4. Derived from the ANSI C standard.

## 11300 NAME

11301       stdint.h — integer types

## 11302 SYNOPSIS

11303       #include &lt;stdint.h&gt;

## 11304 DESCRIPTION

11305 cx       Some of the functionality described on this reference page extends the ISO C standard.  
 11306       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 11307       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
 11308       symbols in this header.

11309       The <stdint.h> header shall declare sets of integer types having specified widths, and shall  
 11310       define corresponding sets of macros. It shall also define macros that specify limits of integer  
 11311       types corresponding to types defined in other standard headers.

11312       **Note:**     The “width” of an integer type is the number of bits used to store its value in a pure binary  
 11313       system; the actual type may use more bits than that (for example, a 28-bit type could be stored  
 11314       in 32 bits of actual storage). An  $N$ -bit signed type has values in the range  $-2^{N-1}$  or  $1-2^{N-1}$  to  
 11315        $2^{N-1}-1$ , while an  $N$ -bit unsigned type has values in the range 0 to  $2^N-1$ .

11316       Types are defined in the following categories:

- 11317       • Integer types having certain exact widths
- 11318       • Integer types having at least certain specified widths
- 11319       • Fastest integer types having at least certain specified widths
- 11320       • Integer types wide enough to hold pointers to objects
- 11321       • Integer types having greatest width

11322       (Some of these types may denote the same type.)

11323       Corresponding macros specify limits of the declared types and construct suitable constants.

11324       For each type described herein that the implementation provides, the <stdint.h> header shall  
 11325       declare that **typedef** name and define the associated macros. Conversely, for each type described  
 11326       herein that the implementation does not provide, the <stdint.h> header shall not declare that  
 11327       **typedef** name, nor shall it define the associated macros. An implementation shall provide those  
 11328       types described as required, but need not provide any of the others (described as optional).

11329       **Integer Types**

11330       When **typedef** names differing only in the absence or presence of the initial  $u$  are defined, they  
 11331       shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999  
 11332       standard, Section 6.2.5; an implementation providing one of these corresponding types shall also  
 11333       provide the other.

11334       In the following descriptions, the symbol  $N$  represents an unsigned decimal integer with no  
 11335       leading zeros (for example, 8 or 24, but not 04 or 048).

- 11336       • Exact-width integer types

11337       The **typedef** name **int $N$ \_t** designates a signed integer type with width  $N$ , no padding bits,  
 11338       and a two's-complement representation. Thus, **int8\_t** denotes a signed integer type with a  
 11339       width of exactly 8 bits.

11340       The **typedef** name **uint $N$ \_t** designates an unsigned integer type with width  $N$ . Thus,  
 11341       **uint24\_t** denotes an unsigned integer type with a width of exactly 24 bits.

11342	CX	The following types are required:
11343		<b>int8_t</b>
11344		<b>int16_t</b>
11345		<b>int32_t</b>
11346		<b>uint8_t</b>
11347		<b>uint16_t</b>
11348		<b>uint32_t</b>
11349		If an implementation provides integer types with width 64 that meet these requirements,
11350		then the following types are required:
11351		<b>int64_t</b>
11352		<b>uint64_t</b>
11353	CX	In particular, this will be the case if any of the following are true:
11354		— The implementation supports the <code>_POSIX_V6_ILP32_OFFBIG</code> programming
11355		environment and the application is being built in the <code>_POSIX_V6_ILP32_OFFBIG</code>
11356		programming environment (see the Shell and Utilities volume of IEEE Std 1003.1-2001,
11357		<i>c99</i> , Programming Environments).
11358		— The implementation supports the <code>_POSIX_V6_LP64_OFF64</code> programming environment
11359		and the application is being built in the <code>_POSIX_V6_LP64_OFF64</code> programming
11360		environment.
11361		— The implementation supports the <code>_POSIX_V6_LPBIG_OFFBIG</code> programming
11362		environment and the application is being built in the <code>_POSIX_V6_LPBIG_OFFBIG</code>
11363		programming environment.
11364		All other types of this form are optional.
11365		• Minimum-width integer types
11366		The <b>typedef</b> name <b>int_leastN_t</b> designates a signed integer type with a width of at least <i>N</i> ,
11367		such that no signed integer type with lesser size has at least the specified width. Thus,
11368		<b>int_least32_t</b> denotes a signed integer type with a width of at least 32 bits.
11369		The <b>typedef</b> name <b>uint_leastN_t</b> designates an unsigned integer type with a width of at least
11370		<i>N</i> , such that no unsigned integer type with lesser size has at least the specified width. Thus,
11371		<b>uint_least16_t</b> denotes an unsigned integer type with a width of at least 16 bits.
11372		The following types are required:
11373		<b>int_least8_t</b>
11374		<b>int_least16_t</b>
11375		<b>int_least32_t</b>
11376		<b>int_least64_t</b>
11377		<b>uint_least8_t</b>
11378		<b>uint_least16_t</b>
11379		<b>uint_least32_t</b>
11380		<b>uint_least64_t</b>
11381		All other types of this form are optional.
11382		• Fastest minimum-width integer types
11383		Each of the following types designates an integer type that is usually fastest to operate with
11384		among all integer types that have at least the specified width.

11385 The designated type is not guaranteed to be fastest for all purposes; if the implementation  
 11386 has no clear grounds for choosing one type over another, it will simply pick some integer  
 11387 type satisfying the signedness and width requirements.

11388 The **typedef** name **int\_fastN\_t** designates the fastest signed integer type with a width of at  
 11389 least *N*. The **typedef** name **uint\_fastN\_t** designates the fastest unsigned integer type with a  
 11390 width of at least *N*.

11391 The following types are required:

11392 **int\_fast8\_t**  
 11393 **int\_fast16\_t**  
 11394 **int\_fast32\_t**  
 11395 **int\_fast64\_t**  
 11396 **uint\_fast8\_t**  
 11397 **uint\_fast16\_t**  
 11398 **uint\_fast32\_t**  
 11399 **uint\_fast64\_t**

11400 All other types of this form are optional.

- 11401 • Integer types capable of holding object pointers

11402 The following type designates a signed integer type with the property that any valid pointer  
 11403 to **void** can be converted to this type, then converted back to a pointer to **void**, and the result  
 11404 will compare equal to the original pointer:

11405 **intptr\_t**

11406 The following type designates an unsigned integer type with the property that any valid  
 11407 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and  
 11408 the result will compare equal to the original pointer:

11409 **uintptr\_t**

11410 XSI On XSI-conformant systems, the **intptr\_t** and **uintptr\_t** types are required; otherwise, they  
 11411 are optional.

- 11412 • Greatest-width integer types

11413 The following type designates a signed integer type capable of representing any value of any  
 11414 signed integer type:

11415 **intmax\_t**

11416 The following type designates an unsigned integer type capable of representing any value of  
 11417 any unsigned integer type:

11418 **uintmax\_t**

11419 These types are required.

11420 **Note:** Applications can test for optional types by using the corresponding limit macro from **Limits of**  
 11421 **Specified-Width Integer Types** (on page 321).

## Limits of Specified-Width Integer Types

The following macros specify the minimum and maximum limits of the types declared in the <stdint.h> header. Each macro name corresponds to a similar type name in **Integer Types** (on page 318).

Each instance of any defined macro shall be replaced by a constant expression suitable for use in #if preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign, except where stated to be exactly the given value.

- Limits of exact-width integer types

- Minimum values of exact-width signed integer types:

- {INTN\_MIN} Exactly  $-(2^{N-1})$

- Maximum values of exact-width signed integer types:

- {INTN\_MAX} Exactly  $2^{N-1} - 1$

- Maximum values of exact-width unsigned integer types:

- {UINTN\_MAX} Exactly  $2^N - 1$

- Limits of minimum-width integer types

- Minimum values of minimum-width signed integer types:

- {INT\_LEASTN\_MIN}  $-(2^{N-1} - 1)$

- Maximum values of minimum-width signed integer types:

- {INT\_LEASTN\_MAX}  $2^{N-1} - 1$

- Maximum values of minimum-width unsigned integer types:

- {UINT\_LEASTN\_MAX}  $2^N - 1$

- Limits of fastest minimum-width integer types

- Minimum values of fastest minimum-width signed integer types:

- {INT\_FASTN\_MIN}  $-(2^{N-1} - 1)$

- Maximum values of fastest minimum-width signed integer types:

- {INT\_FASTN\_MAX}  $2^{N-1} - 1$

- Maximum values of fastest minimum-width unsigned integer types:

- {UINT\_FASTN\_MAX}  $2^N - 1$

- Limits of integer types capable of holding object pointers

- Minimum value of pointer-holding signed integer type:

- {INTPTR\_MIN}  $-(2^{15} - 1)$

- Maximum value of pointer-holding signed integer type:

- {INTPTR\_MAX}  $2^{15} - 1$

- Maximum value of pointer-holding unsigned integer type:

11459 {UINTPTR\_MAX}  $2^{16} - 1$

11460 • Limits of greatest-width integer types

11461 — Minimum value of greatest-width signed integer type:

11462 {INTMAX\_MIN}  $-(2^{63} - 1)$

11463 — Maximum value of greatest-width signed integer type:

11464 {INTMAX\_MAX}  $2^{63} - 1$

11465 — Maximum value of greatest-width unsigned integer type:

11466 {UINTMAX\_MAX}  $2^{64} - 1$

#### 11467 Limits of Other Integer Types

11468 The following macros specify the minimum and maximum limits of integer types corresponding  
11469 to types defined in other standard headers.

11470 Each instance of these macros shall be replaced by a constant expression suitable for use in #if  
11471 preprocessing directives, and this expression shall have the same type as would an expression  
11472 that is an object of the corresponding type converted according to the integer promotions. Its  
11473 implementation-defined value shall be equal to or greater in magnitude (absolute value) than  
11474 the corresponding value given below, with the same sign.

11475 • Limits of **ptrdiff\_t**:

11476 {PTRDIFF\_MIN}  $-65\,535$

11477 {PTRDIFF\_MAX}  $+65\,535$

11478 • Limits of **sig\_atomic\_t**:

11479 {SIG\_ATOMIC\_MIN} See below.

11480 {SIG\_ATOMIC\_MAX} See below.

11481 • Limit of **size\_t**:

11482 {SIZE\_MAX}  $65\,535$

11483 • Limits of **wchar\_t**:

11484 {WCHAR\_MIN} See below.

11485 {WCHAR\_MAX} See below.

11486 • Limits of **wint\_t**:

11487 {WINT\_MIN} See below.

11488 {WINT\_MAX} See below.

11489 If **sig\_atomic\_t** (see the <signal.h> header) is defined as a signed integer type, the value of  
11490 {SIG\_ATOMIC\_MIN} shall be no greater than  $-127$  and the value of {SIG\_ATOMIC\_MAX} shall  
11491 be no less than  $127$ ; otherwise, **sig\_atomic\_t** shall be defined as an unsigned integer type, and the  
11492 value of {SIG\_ATOMIC\_MIN} shall be  $0$  and the value of {SIG\_ATOMIC\_MAX} shall be no less  
11493 than  $255$ .

11494 If **wchar\_t** (see the <stddef.h> header) is defined as a signed integer type, the value of  
11495 {WCHAR\_MIN} shall be no greater than  $-127$  and the value of {WCHAR\_MAX} shall be no less  
11496 than  $127$ ; otherwise, **wchar\_t** shall be defined as an unsigned integer type, and the value of  
11497 {WCHAR\_MIN} shall be  $0$  and the value of {WCHAR\_MAX} shall be no less than  $255$ .



11498 If **wint\_t** (see the <wchar.h> header) is defined as a signed integer type, the value of  
 11499 {WINT\_MIN} shall be no greater than −32 767 and the value of {WINT\_MAX} shall be no less  
 11500 than 32 767; otherwise, **wint\_t** shall be defined as an unsigned integer type, and the value of  
 11501 {WINT\_MIN} shall be 0 and the value of {WINT\_MAX} shall be no less than 65 535.

## 11502 **Macros for Integer Constant Expressions**

11503 The following macros expand to integer constant expressions suitable for initializing objects that  
 11504 have integer types corresponding to types defined in the <stdint.h> header. Each macro name  
 11505 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*  
 11506 *integer types*.

11507 Each invocation of one of these macros shall expand to an integer constant expression suitable  
 11508 for use in **#if** preprocessing directives. The type of the expression shall have the same type as  
 11509 would an expression that is an object of the corresponding type converted according to the  
 11510 integer promotions. The value of the expression shall be that of the argument.

11511 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant  
 11512 with a value that does not exceed the limits for the corresponding type.

### 11513 • Macros for minimum-width integer constant expressions

11514 The macro **INTN\_C(value)** shall expand to an integer constant expression corresponding to  
 11515 the type **int\_leastN\_t**. The macro **UINTN\_C(value)** shall expand to an integer constant  
 11516 expression corresponding to the type **uint\_leastN\_t**. For example, if **uint\_least64\_t** is a name  
 11517 for the type **unsigned long long**, then **UINT64\_C(0x123)** might expand to the integer  
 11518 constant 0x123ULL.

### 11519 • Macros for greatest-width integer constant expressions

11520 The following macro expands to an integer constant expression having the value specified by  
 11521 its argument and the type **intmax\_t**:

11522 **INTMAX\_C(value)**

11523 The following macro expands to an integer constant expression having the value specified by  
 11524 its argument and the type **uintmax\_t**:

11525 **UINTMAX\_C(value)**

## 11526 **APPLICATION USAGE**

11527 None.

## 11528 **RATIONALE**

11529 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in  
 11530 freestanding environments, which might not support the formatted I/O functions. In some  
 11531 environments, if the formatted conversion support is not wanted, using this header instead of  
 11532 the <inttypes.h> header avoids defining such a large number of macros.

11533 As a consequence of adding **int8\_t**, the following are true:

- 11534 • A byte is exactly 8 bits.
- 11535 • {CHAR\_BIT} has the value 8, {SCHAR\_MAX} has the value 127, {SCHAR\_MIN} has the  
 11536 value −127 or −128, and {UCHAR\_MAX} has the value 255.

## 11537 **FUTURE DIRECTIONS**

11538 **typedef** names beginning with **int** or **uint** and ending with **\_t** may be added to the types defined  
 11539 in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **\_MAX**,  
 11540 **\_MIN**, or **\_C** may be added to the macros defined in the <stdint.h> header.

11541 **SEE ALSO**11542       **<inttypes.h>**, **<signal.h>**, **<stddef.h>**, **<wchar.h>**11543 **CHANGE HISTORY**

11544       First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

11545       ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

1

11546 **NAME**

11547       stdio.h — standard buffered input/output

11548 **SYNOPSIS**

11549       #include <stdio.h>

11550 **DESCRIPTION**

11551 CX       Some of the functionality described on this reference page extends the ISO C standard.  
11552       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
11553       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
11554       symbols in this header.

11555       The <stdio.h> header shall define the following macros as positive integer constant expressions:

11556       BUFSIZ               Size of <stdio.h> buffers.

11557       \_IIOFBF             Input/output fully buffered.

11558       \_IIOBFB             Input/output line buffered.

11559       \_IIONBF             Input/output unbuffered.

11560 CX       L\_ctermid       Maximum size of character array to hold *ctermid()* output.

11561       L\_tmpnam            Maximum size of character array to hold *tmpnam()* output.

11562       SEEK\_CUR            Seek relative to current position.

11563       SEEK\_END            Seek relative to end-of-file.

11564       SEEK\_SET            Seek relative to start-of-file.

11565       The following macros shall be defined as positive integer constant expressions which denote  
11566       implementation limits:

11567       {FILENAME\_MAX}       Maximum size in bytes of the longest filename string that the  
11568       implementation guarantees can be opened.

11569       {FOPEN\_MAX}         Number of streams which the implementation guarantees can be open  
11570       simultaneously. The value is at least eight.

11571       {TMP\_MAX}            Minimum number of unique filenames generated by *tmpnam()*.  
11572       Maximum number of times an application can call *tmpnam()* reliably. The  
11573 XSI       value of {TMP\_MAX} is at least 25. On XSI-conformant systems, the  
11574       value of {TMP\_MAX} is at least 10 000.

11575       The following macro name shall be defined as a negative integer constant expression:

11576       EOF                 End-of-file return value.

11577       The following macro name shall be defined as a null pointer constant:

11578       NULL                Null pointer.

11579       The following macro name shall be defined as a string constant:

11580 XSI       P\_tmpdir       Default directory prefix for *tmpnam()*.

11581       The following shall be defined as expressions of type “pointer to **FILE**” that point to the **FILE**  
11582       objects associated, respectively, with the standard error, input, and output streams:

11583       stderr                Standard error output stream.

11584       stdin                Standard input stream.

11585	<b>stdout</b>	Standard output stream.
11586	The following data types shall be defined through <b>typedef</b> :	
11587	<b>FILE</b>	A structure containing information about a file.
11588	<b>fpos_t</b>	A non-array type containing all information needed to specify uniquely
11589		every position within a file.
11590 XSI	<b>va_list</b>	As described in <stdarg.h>.
11591	<b>size_t</b>	As described in <stddef.h>.
11592	The following shall be declared as functions and may also be defined as macros. Function	
11593	prototypes shall be provided.	
11594	void	clearerr(FILE *);
11595 CX	char	*ctermid(char *);
11596	int	fclose(FILE *);
11597 CX	FILE	*fdopen(int, const char *);
11598	int	feof(FILE *);
11599	int	ferror(FILE *);
11600	int	fflush(FILE *);
11601	int	fgetc(FILE *);
11602	int	fgetpos(FILE *restrict, fpos_t *restrict);
11603	char	*fgets(char *restrict, int, FILE *restrict);
11604 CX	int	fileno(FILE *);
11605 TSF	void	flockfile(FILE *);
11606	FILE	*fopen(const char *restrict, const char *restrict);
11607	int	fprintf(FILE *restrict, const char *restrict, ...);
11608	int	fputc(int, FILE *);
11609	int	fputs(const char *restrict, FILE *restrict);
11610	size_t	fread(void *restrict, size_t, size_t, FILE *restrict);
11611	FILE	*freopen(const char *restrict, const char *restrict,
11612		FILE *restrict);
11613	int	fscanf(FILE *restrict, const char *restrict, ...);
11614	int	fseek(FILE *, long, int);
11615 CX	int	fseeko(FILE *, off_t, int);
11616	int	fsetpos(FILE *, const fpos_t *);
11617	long	ftell(FILE *);
11618 CX	off_t	ftello(FILE *);
11619 TSF	int	ftrylockfile(FILE *);
11620	void	funlockfile(FILE *);
11621	size_t	fwrite(const void *restrict, size_t, size_t, FILE *restrict);
11622	int	getc(FILE *);
11623	int	getchar(void);
11624 TSF	int	getc_unlocked(FILE *);
11625	int	getchar_unlocked(void);
11626	char	*gets(char *);
11627 CX	int	pclose(FILE *);
11628	void	perror(const char *);
11629 CX	FILE	*popen(const char *, const char *);
11630	int	printf(const char *restrict, ...);
11631	int	putc(int, FILE *);
11632	int	putchar(int);
11633 TSF		

```

11634     int      putc_unlocked(int, FILE *);
11635     int      putchar_unlocked(int);
11636     int      puts(const char *);
11637     int      remove(const char *);
11638     int      rename(const char *, const char *);
11639     void     rewind(FILE *);
11640     int      scanf(const char *restrict, ...);
11641     void     setbuf(FILE *restrict, char *restrict);
11642     int      setvbuf(FILE *restrict, char *restrict, int, size_t);
11643     int      snprintf(char *restrict, size_t, const char *restrict, ...);
11644     int      sprintf(char *restrict, const char *restrict, ...);
11645     int      sscanf(const char *restrict, const char *restrict, int ...);
11646 XSI     char  *tempnam(const char *, const char *);
11647     FILE     *tmpfile(void);
11648     char     *tmpnam(char *);
11649     int      ungetc(int, FILE *);
11650     int      vfprintf(FILE *restrict, const char *restrict, va_list);
11651     int      vfprintf(FILE *restrict, const char *restrict, va_list);
11652     int      vprintf(const char *restrict, va_list);
11653     int      vscanf(const char *restrict, va_list);
11654     int      vsnprintf(char *restrict, size_t, const char *restrict,
11655                        va_list);
11656     int      vsprintf(char *restrict, const char *restrict, va_list);
11657     int      vsscanf(const char *restrict, const char *restrict,
11658                      va_list arg);

```

11659 XSI      Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.

## 11660 APPLICATION USAGE

11661            None.

## 11662 RATIONALE

11663            None.

## 11664 FUTURE DIRECTIONS

11665            None.

## 11666 SEE ALSO

11667            <stdarg.h>, <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001,  
11668            *clearerr()*, *ctermid()*, *fclose()*, *fdopen()*, *fgetc()*, *fgetpos()*, *ferror()*, *feof()*, *fflush()*, *fgets()*, *fileno()*,  
11669            *flockfile()*, *fopen()*, *fputc()*, *fputs()*, *fread()*, *freopen()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*,  
11670            *getc\_unlocked()*, *getwchar()*, *getchar()*, *getopt()*, *gets()*, *pclose()*, *perror()*, *popen()*, *printf()*, *putc()*,  
11671            *putchar()*, *puts()*, *putwchar()*, *remove()*, *rename()*, *rewind()*, *scanf()*, *setbuf()*, *setvbuf()*, *sscanf()*,  
11672            *stdin*, *system()*, *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vfscanf()*, *vscanf()*, *vprintf()*, *vsscanf()*

## 11673 CHANGE HISTORY

11674            First released in Issue 1. Derived from Issue 1 of the SVID.

## 11675 Issue 5

11676            The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11677            Large File System extensions are added.

11678            The constant *L\_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as  
11679            extensions and LEGACY.

- 11680 The *cuserid()* and *getopt()* functions are marked LEGACY.
- 11681 **Issue 6**
- 11682 The constant *L\_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed  
11683 as they were previously marked LEGACY.
- 11684 The *cuserid()*, *getopt()*, and *getw()* functions are removed as they were previously marked  
11685 LEGACY.
- 11686 Several functions are marked as part of the Thread-Safe Functions option.
- 11687 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the  
11688 description of the **fpos\_t** type is now explicitly updated to exclude array types.
- 11689 Extensions beyond the ISO C standard are marked.

11690 **NAME**

11691        stdlib.h — standard library definitions

11692 **SYNOPSIS**

11693        #include <stdlib.h>

11694 **DESCRIPTION**

11695 CX       Some of the functionality described on this reference page extends the ISO C standard.  
11696       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
11697       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
11698       symbols in this header.

11699       The <stdlib.h> header shall define the following macros:

11700       EXIT\_FAILURE   Unsuccessful termination for *exit()*; evaluates to a non-zero value.

11701       EXIT\_SUCCESS   Successful termination for *exit()*; evaluates to 0.

11702       NULL           Null pointer.

11703       {RAND\_MAX}     Maximum value returned by *rand()*; at least 32 767.

11704       {MB\_CUR\_MAX}   Integer expression whose value is the maximum number of bytes in a  
11705       character specified by the current locale.

11706       The following data types shall be defined through **typedef**:

11707       **div\_t**           Structure type returned by the *div()* function.

11708       **ldiv\_t**          Structure type returned by the *ldiv()* function.

11709       **lldiv\_t**         Structure type returned by the *lldiv()* function.

11710       **size\_t**          As described in <stddef.h>.

11711       **wchar\_t**         As described in <stddef.h>.

11712       In addition, the following symbolic names and macros shall be defined as in <sys/wait.h>, for  
11713       use in decoding the return value from *system()*:

11714 XSI       WNOHANG  
11715       WUNTRACED  
11716       WEXITSTATUS  
11717       WIFEXITED  
11718       WIFSIGNALED  
11719       WIFSTOPPED  
11720       WSTOPSIG  
11721       WTERMSIG  
11722

11723       The following shall be declared as functions and may also be defined as macros. Function  
11724       prototypes shall be provided.

11725       void            \_Exit(int);  
11726 XSI       long        a64l(const char \*);  
11727       void            abort(void);  
11728       int             abs(int);  
11729       int             atexit(void (\*)(void));  
11730       double          atof(const char \*);  
11731       int             atoi(const char \*);  
11732       long            atol(const char \*);

```

11733     long long    atoll(const char *);
11734     void         *bsearch(const void *, const void *, size_t, size_t,
11735                           int (*)(const void *, const void *));
11736     void         *calloc(size_t, size_t);
11737     div_t        div(int, int);
11738 XSI    double    drand48(void);
11739     char         *ecvt(double, int, int *restrict, int *restrict); (LEGACY)
11740     double       erand48(unsigned short[3]);
11741     void         exit(int);
11742 XSI    char         *fcvt(double, int, int *restrict, int *restrict); (LEGACY)
11743     void         free(void *);
11744 XSI    char         *gcvt(double, int, char *); (LEGACY)
11745     char         *getenv(const char *);
11746 XSI    int         getsubopt(char **, char *const *, char **);
11747     int         grantpt(int);
11748     char         *initstate(unsigned, char *, size_t);
11749     long         jrand48(unsigned short[3]);
11750     char         *l64a(long);
11751     long         labs(long);
11752 XSI    void         lcong48(unsigned short[7]);
11753     ldiv_t       ldiv(long, long);
11754     long long    llabs(long long);
11755     lldiv_t      lldiv(long long, long long);
11756 XSI    long         lrand48(void);
11757     void         *malloc(size_t);
11758     int         mblen(const char *, size_t);
11759     size_t       mbstowcs(wchar_t *restrict, const char *restrict, size_t);
11760     int         mbtowc(wchar_t *restrict, const char *restrict, size_t);
11761 XSI    char         *mktemp(char *); (LEGACY)
11762     int         mkstemp(char *);
11763     long         mrand48(void);
11764     long         nrand48(unsigned short[3]);
11765 ADV    int         posix_memalign(void **, size_t, size_t);
11766 XSI    int         posix_openpt(int);
11767     char         *ptsname(int);
11768     int         putenv(char *);
11769     void         qsort(void *, size_t, size_t, int (*)(const void *,
11770                                                         const void *));
11771     int         rand(void);
11772 TSF    int         rand_r(unsigned *);
11773 XSI    long         random(void);
11774     void         *realloc(void *, size_t);
11775 XSI    char         *realpath(const char *restrict, char *restrict);
11776     unsigned short seed48(unsigned short[3]);
11777 CX     int         setenv(const char *, const char *, int);
11778 XSI    void         setkey(const char *);
11779     char         *setstate(const char *);
11780     void         srand(unsigned);
11781 XSI    void         srand48(long);
11782     void         srand48(unsigned);
11783     double       strtod(const char *restrict, char **restrict);
11784     float        strtod(const char *restrict, char **restrict);

```



```

11785      long          strtol(const char *restrict, char **restrict, int);
11786      long double    strtold(const char *restrict, char **restrict);
11787      long long       strtoll(const char *restrict, char **restrict, int);
11788      unsigned long   strtoul(const char *restrict, char **restrict, int);
11789      unsigned long long
11790      strtoull(const char *restrict, char **restrict, int);
11791      int             system(const char *);
11792 XSI      int         unlockpt(int);
11793 CX      int         unsetenv(const char *);
11794      size_t          wcstombs(char *restrict, const wchar_t *restrict, size_t);
11795      int             wctomb(char *, wchar_t);

11796 XSI      Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>,
11797      <limits.h>, <math.h>, and <sys/wait.h>.

```

## 11798 APPLICATION USAGE

11799 None.

## 11800 RATIONALE

11801 None.

## 11802 FUTURE DIRECTIONS

11803 None.

## 11804 SEE ALSO

11805 <limits.h>, <math.h>, <stddef.h>, <sys/types.h>, <sys/wait.h>, the System Interfaces volume of  
11806 IEEE Std 1003.1-2001, *\_Exit()*, *a64l()*, *abort()*, *abs()*, *atexit()*, *atof()*, *atoi()*, *atoll()*, *bsearch()*,  
11807 *calloc()*, *div()*, *drand48()*, *erand48()*, *exit()*, *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*,  
11808 *jrand48()*, *l64a()*, *labs()*, *lcong48()*, *ldiv()*, *llabs()*, *lldiv()*, *lrand48()*, *malloc()*, *mblen()*, *mbstowcs()*,  
11809 *mbtowc()*, *mkstemp()*, *mrand48()*, *nrand48()*, *posix\_memalign()*, *ptsname()*, *putenv()*, *qsort()*,  
11810 *rand()*, *realloc()*, *realpath()*, *setstate()*, *srand()*, *srand48()*, *srandom()*, *strtod()*, *strtof()*, *strtol()*,  
11811 *strtold()*, *strtoll()*, *strtoul()*, *strtoull()*, *unlockpt()*, *wcstombs()*, *wctomb()*

## 11812 CHANGE HISTORY

11813 First released in Issue 3.

### 11814 Issue 5

11815 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11816 The *ttyslot()* and *valloc()* functions are marked LEGACY.

11817 The type of the third argument to *initstate()* is changed from **int** to **size\_t**. The type of the return  
11818 value from *setstate()* is changed from **char** to **char \***, and the type of the first argument is  
11819 changed from **char \*** to **const char \***.

### 11820 Issue 6

11821 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be  
11822 consistent with the reference page.

11823 The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be  
11824 consistent with the reference page.

11825 The *rand\_r()* function is marked as part of the Thread-Safe Functions option.

11826 Function prototypes for *setenv()* and *unsetenv()* are added.

11827 The *posix\_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

11828 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

- 11829 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.
- 11830 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY.
- 11831 Extensions beyond the ISO C standard are marked.

11832 **NAME**

11833       string.h — string operations

11834 **SYNOPSIS**

11835       #include <string.h>

11836 **DESCRIPTION**

11837 CX       Some of the functionality described on this reference page extends the ISO C standard.  
11838       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
11839       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
11840       symbols in this header.

11841       The <string.h> header shall define the following:

11842       NULL           Null pointer constant.

11843       size\_t          As described in <stddef.h>.

11844       The following shall be declared as functions and may also be defined as macros. Function  
11845       prototypes shall be provided.

11846 XSI       void       \*memcpy(void \*restrict, const void \*restrict, int, size\_t);

11847       void       \*memchr(const void \*, int, size\_t);

11848       int        memcmp(const void \*, const void \*, size\_t);

11849       void       \*memcpy(void \*restrict, const void \*restrict, size\_t);

11850       void       \*memmove(void \*, const void \*, size\_t);

11851       void       \*memset(void \*, int, size\_t);

11852       char       \*strcat(char \*restrict, const char \*restrict);

11853       char       \*strchr(const char \*, int);

11854       int        strcmp(const char \*, const char \*);

11855       int        strcoll(const char \*, const char \*);

11856       char       \*strcpy(char \*restrict, const char \*restrict);

11857       size\_t     strcspn(const char \*, const char \*);

11858 XSI       char       \*strdup(const char \*);

11859       char       \*strerror(int);

11860 TSF       int        \*strerror\_r(int, char \*, size\_t);

11861       size\_t     strlen(const char \*);

11862       char       \*strncat(char \*restrict, const char \*restrict, size\_t);

11863       int        strncmp(const char \*, const char \*, size\_t);

11864       char       \*strncpy(char \*restrict, const char \*restrict, size\_t);

11865       char       \*strpbrk(const char \*, const char \*);

11866       char       \*strrchr(const char \*, int);

11867       size\_t     strspn(const char \*, const char \*);

11868       char       \*strstr(const char \*, const char \*);

11869       char       \*strtok(char \*restrict, const char \*restrict);

11870 TSF       char       \*strtok\_r(char \*, const char \*, char \*\*);

11871       size\_t     strxfrm(char \*restrict, const char \*restrict, size\_t);

11872 XSI       Inclusion of the <string.h> header may also make visible all symbols from <stddef.h>.

11873 **APPLICATION USAGE**

11874       None.

11875 **RATIONALE**

11876       None.

11877 **FUTURE DIRECTIONS**

11878       None.

11879 **SEE ALSO**

11880       <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *memcpy()*,  
11881       *memchr()*, *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*,  
11882       *strcpy()*, *strcspn()*, *strdup()*, *strerror()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*,  
11883       *strspn()*, *strstr()*, *strtok()*, *strxfrm()*

11884 **CHANGE HISTORY**

11885       First released in Issue 1. Derived from Issue 1 of the SVID.

11886 **Issue 5**

11887       The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11888 **Issue 6**11889       The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.

11890       This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

11891       The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

11892 **NAME**

11893 strings.h — string operations

11894 **SYNOPSIS**

11895 XSI #include <strings.h>

11896

11897 **DESCRIPTION**

11898 The following shall be declared as functions and may also be defined as macros. Function  
11899 prototypes shall be provided.

11900 int bcmp(const void \*, const void \*, size\_t); (**LEGACY**)

11901 void bcopy(const void \*, void \*, size\_t); (**LEGACY**)

11902 void bzero(void \*, size\_t); (**LEGACY**)

11903 int ffs(int);

11904 char \*index(const char \*, int); (**LEGACY**)

11905 char \*rindex(const char \*, int); (**LEGACY**)

11906 int strcasecmp(const char \*, const char \*);

11907 int strncasecmp(const char \*, const char \*, size\_t);

11908 The `size_t` type shall be defined through `typedef` as described in <stddef.h>.

11909 **APPLICATION USAGE**

11910 None.

11911 **RATIONALE**

11912 None.

11913 **FUTURE DIRECTIONS**

11914 None.

11915 **SEE ALSO**

11916 <stddef.h>, the System Interfaces volume of IEEE Std 1003.1-2001, `ffs()`, `strcasecmp()`,  
11917 `strncasecmp()`

11918 **CHANGE HISTORY**

11919 First released in Issue 4, Version 2.

11920 **Issue 6**

11921 The Open Group Corrigendum U021/2 is applied, correcting the prototype for `index()` to be  
11922 consistent with the reference page.

11923 The `bcmp()`, `bcopy()`, `bzero()`, `index()`, and `rindex()` functions are marked LEGACY.

## 11924 NAME

11925       stropts.h — STREAMS interface (**STREAMS**)

## 11926 SYNOPSIS

11927 XSR     #include &lt;stropts.h&gt;

11928

## 11929 DESCRIPTION

11930       The <stropts.h> header shall define the **bandinfo** structure that includes at least the following  
11931       members:

11932	unsigned char	bi_pri	Priority band.
11933	int	bi_flag	Flushing type.

11934       The <stropts.h> header shall define the **strpeek** structure that includes at least the following  
11935       members:

11936	struct strbuf	ctlbuf	The control portion of the message.
11937	struct strbuf	databuf	The data portion of the message.
11938	t_uscalar_t	flags	RS_HIPRI or 0.

11939       The <stropts.h> header shall define the **strbuf** structure that includes at least the following  
11940       members:

11941	int	maxlen	Maximum buffer length.
11942	int	len	Length of data.
11943	char	*buf	Pointer to buffer.

11944       The <stropts.h> header shall define the **strfdinsert** structure that includes at least the following  
11945       members:

11946	struct strbuf	ctlbuf	The control portion of the message.
11947	struct strbuf	databuf	The data portion of the message.
11948	t_uscalar_t	flags	RS_HIPRI or 0.
11949	int	fildes	File descriptor of the other STREAM.
11950	int	offset	Relative location of the stored value.

11951       The <stropts.h> header shall define the **striocctl** structure that includes at least the following  
11952       members:

11953	int	ic_cmd	<i>ioctl()</i> command.
11954	int	ic_timeout	Timeout for response.
11955	int	ic_len	Length of data.
11956	char	*ic_dp	Pointer to buffer.

11957       The <stropts.h> header shall define the **strrecvfd** structure that includes at least the following  
11958       members:

11959	int	fda	Received file descriptor.
11960	uid_t	uid	UID of sender.
11961	gid_t	gid	GID of sender.

11962       The **uid\_t** and **gid\_t** types shall be defined through **typedef** as described in <sys/types.h>.11963       The <stropts.h> header shall define the **t\_scalar\_t** and **t\_uscalar\_t** types, respectively, as signed  
11964       and unsigned opaque types of equal length of at least 32 bits.11965       The <stropts.h> header shall define the **str\_list** structure that includes at least the following  
11966       members:

11967	int	sl_nmods	Number of STREAMS module names.
11968	struct str_mlist	*sl_modlist	STREAMS module names.
11969	The <stropts.h> header shall define the <b>str_mlist</b> structure that includes at least the following		
11970	member:		
11971	char	l_name[FMNAMESZ+1]	A STREAMS module name.
11972	At least the following macros shall be defined for use as the <i>request</i> argument to <i>ioctl()</i> :		
11973	I_PUSH		Push a STREAMS module.
11974	I_POP		Pop a STREAMS module.
11975	I_LOOK		Get the top module name.
11976	I_FLUSH		Flush a STREAM.
11977	I_FLUSHBAND		Flush one band of a STREAM.
11978	I_SETSIG		Ask for notification signals.
11979	I_GETSIG		Retrieve current notification signals.
11980	I_FIND		Look for a STREAMS module.
11981	I_PEEK		Peek at the top message on a STREAM.
11982	I_SRDOPT		Set the read mode.
11983	I_GRDOPT		Get the read mode.
11984	I_NREAD		Size the top message.
11985	I_FDINSERT		Send implementation-defined information about another STREAM.
11986	I_STR		Send a STREAMS <i>ioctl()</i> .
11987	I_SWROPT		Set the write mode.
11988	I_GWROPT		Get the write mode.
11989	I_SENDFD		Pass a file descriptor through a STREAMS pipe.
11990	I_RECVFD		Get a file descriptor sent via I_SENDFD.
11991	I_LIST		Get all the module names on a STREAM.
11992	I_ATMARK		Is the top message “marked”?
11993	I_CKBAND		See if any messages exist in a band.
11994	I_GETBAND		Get the band of the top message on a STREAM.
11995	I_CANPUT		Is a band writable?
11996	I_SETCLTIME		Set close time delay.
11997	I_GETCLTIME		Get close time delay.
11998	I_LINK		Connect two STREAMs.
11999	I_UNLINK		Disconnect two STREAMs.
12000	I_PLINK		Persistently connect two STREAMs.
12001	I_PUNLINK		Dismantle a persistent STREAMS link.

12002	At least the following macros shall be defined for use with I_LOOK:	
12003	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.
12004	At least the following macros shall be defined for use with I_FLUSH:	
12005	FLUSHR	Flush read queues.
12006	FLUSHW	Flush write queues.
12007	FLUSHRW	Flush read and write queues.
12008	At least the following macros shall be defined for use with I_SETSIG:	
12009	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue.
12010		
12011	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue.
12012		
12013	S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue.
12014		
12015	S_HIPRI	A high-priority message is present on a STREAM head read queue.
12016	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
12017		
12018		
12019	S_WRNORM	Equivalent to S_OUTPUT.
12020	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full.
12021		
12022	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the front of the STREAM head read queue.
12023		
12024	S_ERROR	Notification of an error condition reaches the STREAM head.
12025	S_HANGUP	Notification of a hangup reaches the STREAM head.
12026	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.
12027		
12028		
12029	At least the following macros shall be defined for use with I_PEEK:	
12030	RS_HIPRI	Only look for high-priority messages.
12031	At least the following macros shall be defined for use with I_SRDOPT:	
12032	RNORM	Byte-STREAM mode, the default.
12033	RMSGD	Message-discard mode.
12034	RMSGN	Message-non-discard mode.
12035	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
12036		
12037	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12038	RPROTDIS	Discard the control part of a message, delivering any data part, when a process issues a <i>read()</i> .
12039		



12040 At least the following macros shall be defined for use with I\_SWOPT:

12041 SNDZERO Send a zero-length message downstream when a *write()* of 0 bytes occurs.

12042 At least the following macros shall be defined for use with I\_ATMARK:

12043 ANYMARK Check if the message is marked.

12044 LASTMARK Check if the message is the last one marked on the queue.

12045 At least the following macros shall be defined for use with I\_UNLINK:

12046 MUXID\_ALL Unlink all STREAMs linked to the STREAM associated with *fildev*.

12047 The following macros shall be defined for *getmsg()*, *getpmsg()*, *putmsg()*, and *putpmsg()*:

12048 MSG\_ANY Receive any message.

12049 MSG\_BAND Receive message from specified band.

12050 MSG\_HIPRI Send/receive high-priority message.

12051 MORECTL More control information is left in message.

12052 MOREDATA More data is left in message.

12053 The <stropts.h> header may make visible all of the symbols from <unistd.h>.

12054 The following shall be declared as functions and may also be defined as macros. Function

12055 prototypes shall be provided.

12056 int isastream(int);

12057 int getmsg(int, struct strbuf \*restrict, struct strbuf \*restrict,

12058 int \*restrict);

12059 int getpmsg(int, struct strbuf \*restrict, struct strbuf \*restrict,

12060 int \*restrict, int \*restrict);

12061 int ioctl(int, int, ... );

12062 int putmsg(int, const struct strbuf \*, const struct strbuf \*, int);

12063 int putpmsg(int, const struct strbuf \*, const struct strbuf \*, int,

12064 int);

12065 int fattach(int, const char \*);

12066 int fdetach(const char \*);

12067 **APPLICATION USAGE**

12068 None.

12069 **RATIONALE**

12070 None.

12071 **FUTURE DIRECTIONS**

12072 None.

12073 **SEE ALSO**

12074 <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *close()*, *fcntl()*,

12075 *getmsg()*, *ioctl()*, *open()*, *pipe()*, *read()*, *poll()*, *putmsg()*, *signal()*, *write()*

12076 **CHANGE HISTORY**

12077 First released in Issue 4, Version 2.

12078 **Issue 5**

12079 The *flags* members of the **strpeek** and **strfdinsert** structures are changed from **type long** to  
12080 **t\_uscalar\_t**.

12081 **Issue 6**

12082 This header is marked as part of the XSI STREAMS Option Group.

12083 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.

12084 **NAME**

12085 sys/ipc.h — XSI interprocess communication access structure

12086 **SYNOPSIS**

12087 XSI `#include <sys/ipc.h>`

12088

12089 **DESCRIPTION**

12090 The <sys/ipc.h> header is used by three mechanisms for XSI interprocess communication (IPC):  
 12091 messages, semaphores, and shared memory. All use a common structure type, **ipc\_perm**, to pass  
 12092 information used in determining permission to perform an IPC operation.

12093 The **ipc\_perm** structure shall contain the following members:

12094	<code>uid_t</code>	<code>uid</code>	Owner's user ID.
12095	<code>gid_t</code>	<code>gid</code>	Owner's group ID.
12096	<code>uid_t</code>	<code>cuid</code>	Creator's user ID.
12097	<code>gid_t</code>	<code>cgid</code>	Creator's group ID.
12098	<code>mode_t</code>	<code>mode</code>	Read/write permission.

12099 The **uid\_t**, **gid\_t**, **mode\_t**, and **key\_t** types shall be defined as described in <sys/types.h>.

12100 Definitions shall be provided for the following constants:

12101 Mode bits:

12102	<code>IPC_CREAT</code>	Create entry if key does not exist.
12103	<code>IPC_EXCL</code>	Fail if key exists.
12104	<code>IPC_NOWAIT</code>	Error if request must wait.

12105 Keys:

12106	<code>IPC_PRIVATE</code>	Private key.
-------	--------------------------	--------------

12107 Control commands:

12108	<code>IPC_RMID</code>	Remove identifier.
12109	<code>IPC_SET</code>	Set options.
12110	<code>IPC_STAT</code>	Get options.

12111 The following shall be declared as a function and may also be defined as a macro. A function  
 12112 prototype shall be provided.

12113 `key_t ftok(const char *, int);`

12114 **APPLICATION USAGE**

12115 None.

12116 **RATIONALE**

12117 None.

12118 **FUTURE DIRECTIONS**

12119 None.

12120 **SEE ALSO**

12121 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *ftok()*

12122 **CHANGE HISTORY**

12123 First released in Issue 2. Derived from System V Release 2.0.

12124 **NAME**

12125 sys/mman.h — memory management declarations

12126 **SYNOPSIS**

12127 #include <sys/mman.h>

12128 **DESCRIPTION**

12129 The <sys/mman.h> header shall be supported if the implementation supports at least one of the  
12130 following options:

- 12131 MF • The Memory Mapped Files option
- 12132 SHM • The Shared Memory Objects option
- 12133 ML • The Process Memory Locking option
- 12134 MPR • The Memory Protection option
- 12135 TYM • The Typed Memory Objects option
- 12136 SIO • The Synchronized Input and Output option
- 12137 ADV • The Advisory Information option

1

12138 MC2 If one or more of the Advisory Information, Memory Mapped Files, or Shared Memory Objects  
12139 options are supported, the following protection options shall be defined:

- 12140 MC2 PROT\_READ Page can be read.
- 12141 MC2 PROT\_WRITE Page can be written.
- 12142 MC2 PROT\_EXEC Page can be executed.
- 12143 MC2 PROT\_NONE Page cannot be accessed.

12144 The following *flag* options shall be defined:

- 12145 MF|SHM MAP\_SHARED Share changes.
- 12146 MF|SHM MAP\_PRIVATE Changes are private.
- 12147 MF|SHM MAP\_FIXED Interpret *addr* exactly.

12148 The following flags shall be defined for *msync()*:

- 12149 MF|SIO MS\_ASYNC Perform asynchronous writes.
- 12150 MF|SIO MS\_SYNC Perform synchronous writes.
- 12151 MF|SIO MS\_INVALIDATE Invalidate mappings.

12152 ML The following symbolic constants shall be defined for the *mlockall()* function:

- 12153 ML MCL\_CURRENT Lock currently mapped pages.
- 12154 ML MCL\_FUTURE Lock pages that become mapped.

12155 MF|SHM The symbolic constant MAP\_FAILED shall be defined to indicate a failure from the *mmap()*  
12156 function.

12157 MC1 If the Advisory Information and either the Memory Mapped Files or Shared Memory Objects  
12158 options are supported, values for *advice* used by *posix\_madvise()* shall be defined as follows:

- 12159 POSIX\_MADV\_NORMAL
- 12160 The application has no advice to give on its behavior with respect to the specified range. It
- 12161 is the default characteristic if no advice is given for a range of memory.

12162	POSIX_MADV_SEQUENTIAL	
12163	The application expects to access the specified range sequentially from lower addresses to	
12164	higher addresses.	
12165	POSIX_MADV_RANDOM	
12166	The application expects to access the specified range in a random order.	
12167	POSIX_MADV_WILLNEED	
12168	The application expects to access the specified range in the near future.	
12169	POSIX_MADV_DONTNEED	
12170	The application expects that it will not access the specified range in the near future.	
12171		
12172	TYM	The following flags shall be defined for <i>posix_typed_mem_open()</i> :
12173	POSIX_TYPED_MEM_ALLOCATE	
12174	Allocate on <i>mmap()</i> .	
12175	POSIX_TYPED_MEM_ALLOCATE_CONTIG	
12176	Allocate contiguously on <i>mmap()</i> .	
12177	POSIX_TYPED_MEM_MAP_ALLOCATABLE	
12178	Map on <i>mmap()</i> , without affecting allocatability.	
12179		
12180		The <b>mode_t</b> , <b>off_t</b> , and <b>size_t</b> types shall be defined as described in <sys/types.h>.
12181	TYM	The <sys/mman.h> header shall define the structure <b>posix_typed_mem_info</b> , which includes at
12182		least the following member:
12183	size_t	posix_tmi_length Maximum length which may be allocated
12184		from a typed memory object.
12185		
12186		The following shall be declared as functions and may also be defined as macros. Function
12187		prototypes shall be provided.
12188	MLR	int mlock(const void *, size_t); 1
12189	ML	int mlockall(int); 1
12190	MC3	void *mmap(void *, size_t, int, int, int, off_t); 1
12191	MPR	int mprotect(void *, size_t, int);
12192	MF SIO	int msync(void *, size_t, int);
12193	MLR	int munlock(const void *, size_t); 1
12194	ML	int munlockall(void); 1
12195	MC3	int munmap(void *, size_t); 1
12196	ADV	int posix_madvise(void *, size_t, int);
12197	TYM	int posix_mem_offset(const void *restrict, size_t, off_t *restrict,
12198		size_t *restrict, int *restrict);
12199		int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);
12200		int posix_typed_mem_open(const char *, int, int);
12201	SHM	int shm_open(const char *, int, mode_t);
12202		int shm_unlink(const char *);
12203		

12205               None.

12207               None.

12209               None.

```

12211 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, mlock(), mlockall(),
12212 mmap(), mprotect(), msync(), munlock(), munlockall(), munmap(), posix_mem_offset(),
12213 posix_typed_mem_get_info(), posix_typed_mem_open(), shm_open(), shm_unlink()

```

12215 First released in Issue 4, Version 2.

12217 Updated for alignment with the POSIX Realtime Extension.

12219 The **<sys/mman.h>** header is marked as dependent on support for either the Memory Mapped  
12220 Files, Process Memory Locking, or Shared Memory Objects options.

12221 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The TYM margin code is added to the list of margin codes for the `<sys/mman.h>` header line, as well as for other lines.
  - The `POSIX_TYPED_MEM_ALLOCATE`, `POSIX_TYPED_MEM_ALLOCATE_CONTIG`, and `POSIX_TYPED_MEM_MAP_ALLOCATABLE` flags are added.
  - The `posix_tmi_length` structure is added.
  - The `posix_mem_offset()`, `posix_typed_mem_get_info()`, and `posix_typed_mem_open()` functions are added.

12229 The **restrict** keyword is added to the prototype for *posix\_mem\_offset()*.

12230 IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix\_madvise()*.

12231	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and	1
12232	shading errors for the <i>mlock()</i> and <i>munlock()</i> functions.	1

12233	IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code	1
12234	for the <i>mmap()</i> function from MF SHM to MC3 (notation for MF SHM TYM).	1

12235	IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code	1
12236	for the <i>munmap()</i> function from MF   SHM to MC3 (notation for MF   SHM   TYM).	1

## 12237 NAME

12238 sys/msg.h — XSI message queue structures

## 12239 SYNOPSIS

12240 XSI `#include <sys/msg.h>`

12241

## 12242 DESCRIPTION

12243 The <sys/msg.h> header shall define the following data types through **typedef**:12244 **msgqnum\_t** Used for the number of messages in the message queue.12245 **msglen\_t** Used for the number of bytes allowed in a message queue.12246 These types shall be unsigned integer types that are able to store values at least as large as a type  
12247 **unsigned short**.

12248 The &lt;sys/msg.h&gt; header shall define the following constant as a message operation flag:

12249 MSG\_NOERROR No error if big message.

12250 The **msqid\_ds** structure shall contain the following members:

12251	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
12252	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
12253	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
12254	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <i>msgsnd()</i> .
12255	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <i>msgrcv()</i> .
12256	<code>time_t</code>	<code>msg_stime</code>	Time of last <i>msgsnd()</i> .
12257	<code>time_t</code>	<code>msg_rtime</code>	Time of last <i>msgrcv()</i> .
12258	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

12259 The **pid\_t**, **time\_t**, **key\_t**, **size\_t**, and **ssize\_t** types shall be defined as described in <sys/types.h>.12260 The following shall be declared as functions and may also be defined as macros. Function  
12261 prototypes shall be provided.

12262	<code>int</code>	<code>msgctl(int, int, struct msqid_ds *)</code> ;
12263	<code>int</code>	<code>msgget(key_t, int)</code> ;
12264	<code>ssize_t</code>	<code>msgrcv(int, void *, size_t, long, int)</code> ;
12265	<code>int</code>	<code>msgsnd(int, const void *, size_t, int)</code> ;

12266 In addition, all of the symbols from &lt;sys/ipc.h&gt; shall be defined when this header is included.

## 12267 APPLICATION USAGE

12268 None.

## 12269 RATIONALE

12270 None.

## 12271 FUTURE DIRECTIONS

12272 None.

## 12273 SEE ALSO

12274 <sys/ipc.h>, <sys/types.h>, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

## 12275 CHANGE HISTORY

12276 First released in Issue 2. Derived from System V Release 2.0.



12277 **NAME**

12278 sys/resource.h — definitions for XSI resource operations

12279 **SYNOPSIS**

12280 XSI #include <sys/resource.h>

12281

12282 **DESCRIPTION**

12283 The <sys/resource.h> header shall define the following symbolic constants as possible values of  
12284 the *which* argument of *getpriority()* and *setpriority()*:

12285 PRIO\_PROCESS Identifies the *who* argument as a process ID.

12286 PRIO\_PGRP Identifies the *who* argument as a process group ID.

12287 PRIO\_USER Identifies the *who* argument as a user ID.

12288 The following type shall be defined through **typedef**:

12289 **rlim\_t** Unsigned integer type used for limit values.

12290 The following symbolic constants shall be defined:

12291 RLIM\_INFINITY A value of **rlim\_t** indicating no limit.

12292 RLIM\_SAVED\_MAX A value of type **rlim\_t** indicating an unrepresentable saved hard  
12293 limit.

12294 RLIM\_SAVED\_CUR A value of type **rlim\_t** indicating an unrepresentable saved soft limit.

12295 On implementations where all resource limits are representable in an object of type **rlim\_t**,  
12296 RLIM\_SAVED\_MAX and RLIM\_SAVED\_CUR need not be distinct from RLIM\_INFINITY.

12297 The following symbolic constants shall be defined as possible values of the *who* parameter of  
12298 *getrusage()*:

12299 RUSAGE\_SELF Returns information about the current process.

12300 RUSAGE\_CHILDREN Returns information about children of the current process.

12301 The <sys/resource.h> header shall define the **rlimit** structure that includes at least the following  
12302 members:

12303 rlim\_t rlim\_cur The current (soft) limit.

12304 rlim\_t rlim\_max The hard limit.

12305 The <sys/resource.h> header shall define the **rusage** structure that includes at least the following  
12306 members:

12307 struct timeval ru\_utime User time used.

12308 struct timeval ru\_stime System time used.

12309 The **timeval** structure shall be defined as described in <sys/time.h>.

12310 The following symbolic constants shall be defined as possible values for the *resource* argument of  
12311 *getrlimit()* and *setrlimit()*:

12312 RLIMIT\_CORE Limit on size of **core** file.

12313 RLIMIT\_CPU Limit on CPU time per process.

12314 RLIMIT\_DATA Limit on data segment size.

12315 RLIMIT\_FSIZE Limit on file size.

12316       RLIMIT\_NOFILE       Limit on number of open files.

12317       RLIMIT\_STACK       Limit on stack size.

12318       RLIMIT\_AS       Limit on address space size.

12319       The following shall be declared as functions and may also be defined as macros. Function  
12320       prototypes shall be provided.

12321       int   getpriority(int, id\_t);

12322       int   getrlimit(int, struct rlimit \*);

12323       int   getrusage(int, struct rusage \*);

12324       int   setpriority(int, id\_t, int);

12325       int   setrlimit(int, const struct rlimit \*);

12326       The **id\_t** type shall be defined through **typedef** as described in <sys/types.h>.

12327       Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.

12328 **APPLICATION USAGE**

12329       None.

12330 **RATIONALE**

12331       None.

12332 **FUTURE DIRECTIONS**

12333       None.

12334 **SEE ALSO**

12335       <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *getpriority()*,  
12336       *getrusage()*, *getrlimit()*

12337 **CHANGE HISTORY**

12338       First released in Issue 4, Version 2.

12339 **Issue 5**

12340       Large File System extensions are added.

12341 **NAME**

12342 sys/select.h — select types

12343 **SYNOPSIS**

12344 #include <sys/select.h>

12345 **DESCRIPTION**

12346 The <sys/select.h> header shall define the **timeval** structure that includes at least the following  
12347 members:

12348	time_t	tv_sec	Seconds.
12349	suseconds_t	tv_usec	Microseconds.

12350 The **time\_t** and **suseconds\_t** types shall be defined as described in <sys/types.h>.

12351 The **sigset\_t** type shall be defined as described in <signal.h>.

12352 The **timespec** structure shall be defined as described in <time.h>.

12353 The <sys/select.h> header shall define the **fd\_set** type as a structure.

12354 Each of the following may be declared as a function, or defined as a macro, or both:

12355 **void** *FD\_CLR*(int *fd*, **fd\_set** \**fdset*)

12356 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12357 **int** *FD\_ISSET*(int *fd*, **fd\_set** \**fdset*)

12358 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by  
12359 *fdset*, and 0 otherwise.

12360 **void** *FD\_SET*(int *fd*, **fd\_set** \**fdset*)

12361 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12362 **void** *FD\_ZERO*(**fd\_set** \**fdset*)

12363 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

12364 If implemented as macros, these may evaluate their arguments more than once, so applications  
12365 should ensure that the arguments they supply are never expressions with side effects.

12366 The following shall be defined as a macro:

12367 **FD\_SETSIZE**

12368 Maximum number of file descriptors in an **fd\_set** structure.

12369 The following shall be declared as functions and may also be defined as macros. Function  
12370 prototypes shall be provided.

```
12371 int pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12372             const struct timespec *restrict, const sigset_t *restrict);
12373 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12374            struct timeval *restrict);
```

12375 Inclusion of the <sys/select.h> header may make visible all symbols from the headers  
12376 <signal.h>, <sys/time.h>, and <time.h>.

12377 **APPLICATION USAGE**

12378       None.

12379 **RATIONALE**

12380       None.

12381 **FUTURE DIRECTIONS**

12382       None.

12383 **SEE ALSO**

12384       <signal.h>, <sys/time.h>, <sys/types.h>, <time.h>, the System Interfaces volume of  
12385       IEEE Std 1003.1-2001, *pselect()*, *select()*

12386 **CHANGE HISTORY**

12387       First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

12388       The requirement for the **fd\_set** structure to have a member *fds\_bits* has been removed as per The  
12389       Open Group Base Resolution bwg2001-005.

12390 **NAME**

12391 sys/sem.h — XSI semaphore facility

12392 **SYNOPSIS**

12393 XSI #include <sys/sem.h>

12394

12395 **DESCRIPTION**

12396 The <sys/sem.h> header shall define the following constants and structures.

12397 Semaphore operation flags:

12398 SEM\_UNDO Set up adjust on exit entry.

12399 Command definitions for the *semctl()* function shall be provided as follows:

12400 GETNCNT Get *semncnt*.

12401 GETPID Get *sempid*.

12402 GETVAL Get *semval*.

12403 GETALL Get all cases of *semval*.

12404 GETZCNT Get *semzcnt*.

12405 SETVAL Set *semval*.

12406 SETALL Set all cases of *semval*.

12407 The **semid\_ds** structure shall contain the following members:

12408 struct ipc\_perm sem\_perm Operation permission structure.

12409 unsigned short sem\_nsems Number of semaphores in set.

12410 time\_t sem\_otime Last *semop()* time.

12411 time\_t sem\_ctime Last time changed by *semctl()*.

12412 The **pid\_t**, **time\_t**, **key\_t**, and **size\_t** types shall be defined as described in <sys/types.h>.

12413 A semaphore shall be represented by an anonymous structure containing the following  
12414 members:

12415 unsigned short semval Semaphore value.

12416 pid\_t sempid Process ID of last operation.

12417 unsigned short semncnt Number of processes waiting for *semval*  
12418 to become greater than current value.

12419 unsigned short semzcnt Number of processes waiting for *semval*  
12420 to become 0.

12421 The **sembuf** structure shall contain the following members:

12422 unsigned short sem\_num Semaphore number.

12423 short sem\_op Semaphore operation.

12424 short sem\_flg Operation flags.

12425 The following shall be declared as functions and may also be defined as macros. Function  
12426 prototypes shall be provided.

12427 int semctl(int, int, int, ...);

12428 int semget(key\_t, int, int);

12429 int semop(int, struct sembuf \*, size\_t);

12430           In addition, all of the symbols from **<sys/ipc.h>** shall be defined when this header is included.

12431 **APPLICATION USAGE**

12432           None.

12433 **RATIONALE**

12434           None.

12435 **FUTURE DIRECTIONS**

12436           None.

12437 **SEE ALSO**

12438           **<sys/ipc.h>**, **<sys/types.h>**, *semctl()*, *semget()*, *semop()*

12439 **CHANGE HISTORY**

12440           First released in Issue 2. Derived from System V Release 2.0.

12441 **NAME**

12442 sys/shm.h — XSI shared memory facility

12443 **SYNOPSIS**

12444 XSI `#include <sys/shm.h>`

12445

12446 **DESCRIPTION**

12447 The <sys/shm.h> header shall define the following symbolic constants:

12448 SHM\_RDONLY Attach read-only (else read-write).

12449 SHM\_RND Round attach address to SHMLBA.

12450 The <sys/shm.h> header shall define the following symbolic value:

12451 SHMLBA Segment low boundary address multiple.

12452 The following data types shall be defined through **typedef**:

12453 **shmatt\_t** Unsigned integer used for the number of current attaches that must be able to  
12454 store values at least as large as a type **unsigned short**.

12455 The **shmid\_ds** structure shall contain the following members:

12456	struct ipc_perm	shm_perm	Operation permission structure.
12457	size_t	shm_segsz	Size of segment in bytes.
12458	pid_t	shm_lpid	Process ID of last shared memory operation.
12459	pid_t	shm_cpid	Process ID of creator.
12460	shmatt_t	shm_nattch	Number of current attaches.
12461	time_t	shm_atime	Time of last <i>shmat()</i> .
12462	time_t	shm_dtime	Time of last <i>shmdt()</i> .
12463	time_t	shm_ctime	Time of last change by <i>shmctl()</i> .

12464 The **pid\_t**, **time\_t**, **key\_t**, and **size\_t** types shall be defined as described in <sys/types.h>.

12465 The following shall be declared as functions and may also be defined as macros. Function  
12466 prototypes shall be provided.

```
12467 void *shmat(int, const void *, int);
12468 int shmctl(int, int, struct shmid_ds *);
12469 int shmdt(const void *);
12470 int shmget(key_t, size_t, int);
```

12471 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12472 **APPLICATION USAGE**

12473 None.

12474 **RATIONALE**

12475 None.

12476 **FUTURE DIRECTIONS**

12477 None.

12478 **SEE ALSO**

12479 <sys/ipc.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *shmat()*,  
12480 *shmctl()*, *shmdt()*, *shmget()*

12481 **CHANGE HISTORY**

12482       First released in Issue 2. Derived from System V Release 2.0.

12483 **Issue 5**

12484       The type of *shm\_segsz* is changed from **int** to **size\_t**.



12485 **NAME**

12486 sys/socket.h — main sockets header

12487 **SYNOPSIS**

12488 #include <sys/socket.h>

12489 **DESCRIPTION**

12490 The <sys/socket.h> header shall define the type **socklen\_t**, which is an integer type of width of  
12491 at least 32 bits; see APPLICATION USAGE.

12492 The <sys/socket.h> header shall define the unsigned integer type **sa\_family\_t**.

12493 The <sys/socket.h> header shall define the **sockaddr** structure that includes at least the  
12494 following members:

12495	sa_family_t	sa_family	Address family.
12496	char	sa_data[]	Socket address (variable-length data).

12497 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,  
12498 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.

12499 The <sys/socket.h> header shall define the **sockaddr\_storage** structure. This structure shall be:

- 12500 • Large enough to accommodate all supported protocol-specific address structures
- 12501 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to protocol-  
12502 specific address structures and used to access the fields of those structures without  
12503 alignment problems

12504 The **sockaddr\_storage** structure shall contain at least the following members:

12505	sa_family_t	ss_family
-------	-------------	-----------

12506 When a **sockaddr\_storage** structure is cast as a **sockaddr** structure, the *ss\_family* field of the  
12507 **sockaddr\_storage** structure shall map onto the *sa\_family* field of the **sockaddr** structure. When a  
12508 **sockaddr\_storage** structure is cast as a protocol-specific address structure, the *ss\_family* field  
12509 shall map onto a field of that structure that is of type **sa\_family\_t** and that identifies the  
12510 protocol's address family.

12511 The <sys/socket.h> header shall define the **msghdr** structure that includes at least the following  
12512 members:

12513	void	*msg_name	Optional address.
12514	socklen_t	msg_namelen	Size of address.
12515	struct iovec	*msg_iov	Scatter/gather array.
12516	int	msg_iovlen	Members in <i>msg_iov</i> .
12517	void	*msg_control	Ancillary data; see below.
12518	socklen_t	msg_controllen	Ancillary data buffer <i>len</i> .
12519	int	msg_flags	Flags on received message.

12520 The **msghdr** structure is used to minimize the number of directly supplied parameters to the  
12521 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the  
12522 *recvmsg()* function and *value* only for the *sendmsg()* function.

12523 The **iovec** structure shall be defined as described in <sys/uio.h>.

12524 The <sys/socket.h> header shall define the **cmsg\_hdr** structure that includes at least the following  
12525 members:

12526	socklen_t	cmsg_len	Data byte count, including the <b>cmsg_hdr</b> .
12527	int	cmsg_level	Originating protocol.

12528        `int`            `cmsg_type`     Protocol-specific type.

12529        The **cmsghdr** structure is used for storage of ancillary data object information.

12530        Ancillary data consists of a sequence of pairs, each consisting of a **cmsghdr** structure followed  
12531        by a data array. The data array contains the ancillary data message, and the **cmsghdr** structure  
12532        contains descriptive information that allows an application to correctly parse the data.

12533        The values for `cmsg_level` shall be legal values for the `level` argument to the `getsockopt()` and  
12534        `setsockopt()` functions. The system documentation shall specify the `cmsg_type` definitions for the  
12535        supported protocols.

12536        Ancillary data is also possible at the socket level. The <sys/socket.h> header defines the  
12537        following macro for use as the `cmsg_type` value when `cmsg_level` is `SOL_SOCKET`:

12538        `SCM_RIGHTS`        Indicates that the data array contains the access rights to be sent or  
12539        received.

12540        The <sys/socket.h> header defines the following macros to gain access to the data arrays in the  
12541        ancillary data associated with a message header:

12542        `MSG_DATA(msg)`  
12543        If the argument is a pointer to a **cmsghdr** structure, this macro shall return an unsigned  
12544        character pointer to the data array associated with the **cmsghdr** structure.

12545        `MSG_NXTHDR(mhdr,msg)`  
12546        If the first argument is a pointer to a **msghdr** structure and the second argument is a pointer  
12547        to a **cmsghdr** structure in the ancillary data pointed to by the `msg_control` field of that  
12548        **msghdr** structure, this macro shall return a pointer to the next **cmsghdr** structure, or a null  
12549        pointer if this structure is the last **cmsghdr** in the ancillary data.

12550        `MSG_FIRSTHDR(mhdr)`  
12551        If the argument is a pointer to a **msghdr** structure, this macro shall return a pointer to the  
12552        first **cmsghdr** structure in the ancillary data associated with this **msghdr** structure, or a null  
12553        pointer if there is no ancillary data associated with the **msghdr** structure.

12554        The <sys/socket.h> header shall define the **linger** structure that includes at least the following  
12555        members:

12556        `int`    `l_onoff`     Indicates whether linger option is enabled.  
12557        `int`    `l_linger`    Linger time, in seconds.

12558        The <sys/socket.h> header shall define the following macros, with distinct integer values:

12559        `SOCK_DGRAM`        Datagram socket.

12560 RS     `SOCK_RAW`            Raw Protocol Interface.

12561        `SOCK_SEQPACKET`    Sequenced-packet socket.

12562        `SOCK_STREAM`        Byte-stream socket.

12563        The <sys/socket.h> header shall define the following macro for use as the `level` argument of  
12564        `setsockopt()` and `getsockopt()`.

12565        `SOL_SOCKET`        Options to be accessed at socket level, not protocol level.

12566        The <sys/socket.h> header shall define the following macros, with distinct integer values, for  
12567        use as the `option_name` argument in `getsockopt()` or `setsockopt()` calls:

12568        `SO_ACCEPTCONN`     Socket is accepting connections.

12569	SO_BROADCAST	Transmission of broadcast messages is supported.
12570	SO_DEBUG	Debugging information is being recorded.
12571	SO_DONTROUTE	Bypass normal routing.
12572	SO_ERROR	Socket error status.
12573	SO_KEEPALIVE	Connections are kept alive with periodic messages.
12574	SO_LINGER	Socket lingers on close.
12575	SO_OOBINLINE	Out-of-band data is transmitted in line.
12576	SO_RCVBUF	Receive buffer size.
12577	SO_RCVLOWAT	Receive “low water mark”.
12578	SO_RCVTIMEO	Receive timeout.
12579	SO_REUSEADDR	Reuse of local addresses is supported.
12580	SO_SNDBUF	Send buffer size.
12581	SO_SNDLOWAT	Send “low water mark”.
12582	SO_SNDTIMEO	Send timeout.
12583	SO_TYPE	Socket type.
12584	The <sys/socket.h> header shall define the following macro as the maximum <i>backlog</i> queue length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
12585		
12586	SOMAXCONN	The maximum <i>backlog</i> queue length.
12587	The <sys/socket.h> header shall define the following macros, with distinct integer values, for use as the valid values for the <i>msg_flags</i> field in the <b>msghdr</b> structure, or the <i>flags</i> parameter in <i>recvfrom()</i> , <i>recvmsg()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
12588		
12589		
12590	MSG_CTRUNC	Control data truncated.
12591	MSG_DONTROUTE	Send without using routing tables.
12592	MSG_EOR	Terminates a record (if supported by the protocol).
12593	MSG_OOB	Out-of-band data.
12594	MSG_PEEK	Leave received data in queue.
12595	MSG_TRUNC	Normal data truncated.
12596	MSG_WAITALL	Attempt to fill the read buffer.
12597	The <sys/socket.h> header shall define the following macros, with distinct integer values:	
12598	AF_INET	Internet domain sockets for use with IPv4 addresses.
12599	IP6 AF_INET6	Internet domain sockets for use with IPv6 addresses.
12600	AF_UNIX	UNIX domain sockets.
12601	AF_UNSPEC	Unspecified.
12602	The <sys/socket.h> header shall define the following macros, with distinct integer values:	
12603	SHUT_RD	Disables further receive operations.

```

12604      SHUT_RDWR          Disables further send and receive operations.
12605      SHUT_WR           Disables further send operations.
12606      The following shall be declared as functions and may also be defined as macros. Function
12607      prototypes shall be provided.
12608      int      accept(int, struct sockaddr *restrict, socklen_t *restrict);
12609      int      bind(int, const struct sockaddr *, socklen_t);
12610      int      connect(int, const struct sockaddr *, socklen_t);
12611      int      getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
12612      int      getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
12613      int      getsockopt(int, int, int, void *restrict, socklen_t *restrict);
12614      int      listen(int, int);
12615      ssize_t  recv(int, void *, size_t, int);
12616      ssize_t  recvfrom(int, void *restrict, size_t, int,
12617                        struct sockaddr *restrict, socklen_t *restrict);
12618      ssize_t  recvmsg(int, struct msghdr *, int);
12619      ssize_t  send(int, const void *, size_t, int);
12620      ssize_t  sendmsg(int, const struct msghdr *, int);
12621      ssize_t  sendto(int, const void *, size_t, int, const struct sockaddr *,
12622                     socklen_t);
12623      int      setsockopt(int, int, int, const void *, socklen_t);
12624      int      shutdown(int, int);
12625      int      socket(int, int, int);
12626      int      socketatmark(int);
12627      int      socketpair(int, int, int, int[2]);

```

12628 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/uio.h>.

#### 12629 APPLICATION USAGE

12630 To forestall portability problems, it is recommended that applications not use values larger than  
12631  $2^{31} - 1$  for the **socklen\_t** type.

12632 The **sockaddr\_storage** structure solves the problem of declaring storage for automatic variables  
12633 which is both large enough and aligned enough for storing the socket address data structure of  
12634 any family. For example, code with a file descriptor and without the context of the address  
12635 family can pass a pointer to a variable of this type, where a pointer to a socket address structure  
12636 is expected in calls such as *getpeername()*, and determine the address family by accessing the  
12637 received content after the call.

12638 The example below illustrates a data structure which aligns on a 64-bit boundary. An  
12639 implementation-defined field *\_ss\_align* following *\_ss\_pad1* is used to force a 64-bit alignment  
12640 which covers proper alignment good enough for needs of at least **sockaddr\_in6** (IPv6) and  
12641 **sockaddr\_in** (IPv4) address data structures. The size of padding field *\_ss\_pad1* depends on the  
12642 chosen alignment boundary. The size of padding field *\_ss\_pad2* depends on the value of overall  
12643 size chosen for the total size of the structure. This size and alignment are represented in the  
12644 above example by implementation-defined (not required) constants *\_SS\_MAXSIZE* (chosen  
12645 value 128) and *\_SS\_ALIGNMENT* (with chosen value 8). Constants *\_SS\_PAD1SIZE* (derived  
12646 value 6) and *\_SS\_PAD2SIZE* (derived value 112) are also for illustration and not required. The  
12647 implementation-defined definitions and structure field names above start with an underscore to  
12648 denote implementation private name space. Portable code is not expected to access or reference  
12649 those fields or constants.

```

12650      /*
12651      *   Desired design of maximum size and alignment.

```

```

12652     */
12653 #define _SS_MAXSIZE 128
12654     /* Implementation-defined maximum size. */
12655 #define _SS_ALIGNSIZE (sizeof(int64_t))
12656     /* Implementation-defined desired alignment. */
12657
12658     /*
12659     * Definitions used for sockaddr_storage structure paddings design.
12660     */
12661 #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
12662 #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+ \
12663     _SS_PAD1SIZE + _SS_ALIGNSIZE))
12664 struct sockaddr_storage {
12665     sa_family_t ss_family; /* Address family. */
12666     /*
12667     * Following fields are implementation-defined.
12668     */
12669     char _ss_pad1[_SS_PAD1SIZE];
12670     /* 6-byte pad; this is to make implementation-defined
12671     pad up to alignment field that follows explicit in
12672     the data structure. */
12673     int64_t _ss_align; /* Field to force desired structure
12674     storage alignment. */
12675     char _ss_pad2[_SS_PAD2SIZE];
12676     /* 112-byte pad to achieve desired size,
12677     _SS_MAXSIZE value minus size of ss_family
12678     __ss_pad1, __ss_align fields is 112. */
12679 };
12680
12681 RATIONALE
12682     None.
12683
12684 FUTURE DIRECTIONS
12685     None.
12686
12687 SEE ALSO
12688     <sys/uio.h>, the System Interfaces volume of IEEE Std 1003.1-2001, accept(), bind(), connect(),
12689     getpeername(), getsockname(), getsockopt(), listen(), recv(), recvfrom(), recvmsg(), send(),
12690     sendmsg(), sendto(), setsockopt(), shutdown(), socket(), socketpair()
12691
12692 CHANGE HISTORY
12693     First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
12694
12695     The restrict keyword is added to the prototypes for accept(), getpeername(), getsockname(),
12696     getsockopt(), and recvfrom().

```

## 12691 NAME

12692 sys/stat.h — data returned by the stat() function

## 12693 SYNOPSIS

12694 #include &lt;sys/stat.h&gt;

## 12695 DESCRIPTION

12696 The <sys/stat.h> header shall define the structure of the data returned by the functions *fstat()*,  
12697 *lstat()*, and *stat()*.12698 The **stat** structure shall contain at least the following members:

12699	dev_t	st_dev	Device ID of device containing file.
12700	ino_t	st_ino	File serial number.
12701	mode_t	st_mode	Mode of file (see below).
12702	nlink_t	st_nlink	Number of hard links to the file.
12703	uid_t	st_uid	User ID of file.
12704	gid_t	st_gid	Group ID of file.
12705 XSI	dev_t	st_rdev	Device ID (if file is character or block special).
12706	off_t	st_size	For regular files, the file size in bytes.
12707			For symbolic links, the length in bytes of the
12708			pathname contained in the symbolic link.
12709 SHM			For a shared memory object, the length in bytes.
12710 TYM			For a typed memory object, the length in bytes.
12711			For other file types, the use of this field is
12712			unspecified.
12713	time_t	st_atime	Time of last access.
12714	time_t	st_mtime	Time of last data modification.
12715	time_t	st_ctime	Time of last status change.
12716 XSI	blksize_t	st_blksize	A file system-specific preferred I/O block size for
12717			this object. In some file system types, this may
12718			vary from file to file.
12719	blkcnt_t	st_blocks	Number of blocks allocated for this object.
12720			

12721 The *st\_ino* and *st\_dev* fields taken together uniquely identify the file within the system. The  
 12722 **blkcnt\_t**, **blksize\_t**, **dev\_t**, **ino\_t**, **mode\_t**, **nlink\_t**, **uid\_t**, **gid\_t**, **off\_t**, and **time\_t** types shall be  
 12723 defined as described in <sys/types.h>. The **timespec** structure may be defined as described in 2  
 12724 <time.h>. Times shall be given in seconds since the Epoch. 2

12725 Unless otherwise specified, the structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*,  
 12726 *st\_ctime*, and *st\_mtime* shall have meaningful values for all file types defined in  
 12727 IEEE Std 1003.1-2001.

12728 For symbolic links, the *st\_mode* member shall contain meaningful information, which can be  
 12729 used with the file type macros described below, that take a *mode* argument. The *st\_size* member  
 12730 shall contain the length, in bytes, of the pathname contained in the symbolic link. File mode bits  
 12731 and the contents of the remaining members of the **stat** structure are unspecified. The value  
 12732 returned in the *st\_size* field shall be the length of the contents of the symbolic link, and shall not  
 12733 count a trailing null if one is present.

12734 The following symbolic names for the values of type **mode\_t** shall also be defined.

12735 File type:

12736 XSI **S\_IFMT** Type of file.

12737		S_IFBLK	Block special.
12738		S_IFCHR	Character special.
12739		S_IFIFO	FIFO special.
12740		S_IFREG	Regular.
12741		S_IFDIR	Directory.
12742		S_IFLNK	Symbolic link.
12743		S_IFSOCK	Socket.
12744	File mode bits:		
12745	S_IRWXU	Read, write, execute/search by owner.	
12746		S_IRUSR	Read permission, owner.
12747		S_IWUSR	Write permission, owner.
12748		S_IXUSR	Execute/search permission, owner.
12749	S_IRWXG	Read, write, execute/search by group.	
12750		S_IRGRP	Read permission, group.
12751		S_IWGRP	Write permission, group.
12752		S_IXGRP	Execute/search permission, group.
12753	S_IRWXO	Read, write, execute/search by others.	
12754		S_IROTH	Read permission, others.
12755		S_IWOTH	Write permission, others.
12756		S_IXOTH	Execute/search permission, others.
12757	S_ISUID	Set-user-ID on execution.	
12758	S_ISGID	Set-group-ID on execution.	
12759	XSI	S_ISVTX	On directories, restricted deletion flag.
12760	The bits defined by S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH,		
12761	XSI	S_IWOTH, S_IXOTH, S_ISUID, S_ISGID, and S_ISVTX shall be unique.	
12762	S_IRWXU is the bitwise-inclusive OR of S_IRUSR, S_IWUSR, and S_IXUSR.		
12763	S_IRWXG is the bitwise-inclusive OR of S_IRGRP, S_IWGRP, and S_IXGRP.		
12764	S_IRWXO is the bitwise-inclusive OR of S_IROTH, S_IWOTH, and S_IXOTH.		
12765	Implementations may OR other implementation-defined bits into S_IRWXU, S_IRWXG, and		
12766	S_IRWXO, but they shall not overlap any of the other bits defined in this volume of		
12767	IEEE Std 1003.1-2001. The <i>file permission bits</i> are defined to be those corresponding to the		
12768	bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO.		
12769	The following macros shall be provided to test whether a file is of the specified type. The value		
12770	<i>m</i> supplied to the macros is the value of <i>st_mode</i> from a <b>stat</b> structure. The macro shall evaluate		
12771	to a non-zero value if the test is true; 0 if the test is false.		
12772	S_ISBLK( <i>m</i> )	Test for a block special file.	

12773 S\_ISCHR(*m*) Test for a character special file.  
 12774 S\_ISDIR(*m*) Test for a directory.  
 12775 S\_ISFIFO(*m*) Test for a pipe or FIFO special file.  
 12776 S\_ISREG(*m*) Test for a regular file.  
 12777 S\_ISLNK(*m*) Test for a symbolic link.  
 12778 S\_ISSOCK(*m*) Test for a socket.

12779 The implementation may implement message queues, semaphores, or shared memory objects as  
 12780 distinct file types. The following macros shall be provided to test whether a file is of the  
 12781 specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat**  
 12782 structure. The macro shall evaluate to a non-zero value if the specified object is implemented as  
 12783 a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*.  
 12784 Otherwise, the macro shall evaluate to zero.

12785 S\_TYPEISMQ(*buf*) Test for a message queue.  
 12786 S\_TYPEISSEM(*buf*) Test for a semaphore.  
 12787 S\_TYPEISSHM(*buf*) Test for a shared memory object.

12788 TYP The implementation may implement typed memory objects as distinct file types, and the  
 12789 following macro shall test whether a file is of the specified type. The value of the *buf* argument  
 12790 supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero  
 12791 value if the specified object is implemented as a distinct file type and the specified file type is  
 12792 contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12793 S\_TYPEISTMO(*buf*) Test macro for a typed memory object.  
 12794

12795 The following shall be declared as functions and may also be defined as macros. Function  
 12796 prototypes shall be provided.

```
12797 int    chmod(const char *, mode_t);
12798 int    fchmod(int, mode_t);
12799 int    fstat(int, struct stat *);
12800 int    lstat(const char *restrict, struct stat *restrict);
12801 int    mkdir(const char *, mode_t);
12802 int    mkfifo(const char *, mode_t);
12803 XSI int    mknod(const char *, mode_t, dev_t);
12804 int    stat(const char *restrict, struct stat *restrict);
12805 mode_t umask(mode_t);
```

## 12806 APPLICATION USAGE

12807 Use of the macros is recommended for determining the type of a file.

## 12808 RATIONALE

12809 A conforming C-language application must include <sys/stat.h> for functions that have  
 12810 arguments or return values of type **mode\_t**, so that symbolic values for that type can be used.  
 12811 An alternative would be to require that these constants are also defined by including  
 12812 <sys/types.h>.

12813 The S\_ISUID and S\_ISGID bits may be cleared on any write, not just on *open()*, as some historical  
 12814 implementations do.

12815 System calls that update the time entry fields in the **stat** structure must be documented by the  
 12816 implementors. POSIX-conforming systems should not update the time entry fields for functions



12817	listed in the System Interfaces volume of IEEE Std 1003.1-2001 unless the standard requires that	
12818	they do, except in the case of documented extensions to the standard.	
12819	Note that <i>st_dev</i> must be unique within a Local Area Network (LAN) in a “system” made up of	
12820	multiple computers’ file systems connected by a LAN.	
12821	Networked implementations of a POSIX-conforming system must guarantee that all files visible	
12822	within the file tree (including parts of the tree that may be remotely mounted from other	
12823	machines on the network) on each individual processor are uniquely identified by the	
12824	combination of the <i>st_ino</i> and <i>st_dev</i> fields.	
12825	The unit for the <i>st_blocks</i> member of the <b>stat</b> structure is not defined within IEEE Std 1003.1-2001.	1
12826	In some implementations it is 512 bytes. It may differ on a file system basis. There is no	1
12827	correlation between values of the <i>st_blocks</i> and <i>st_blksize</i> , and the <i>f_bsize</i> (from <sys/statvfs.h>)	1
12828	structure members.	1
12829	Traditionally, some implementations defined the multiplier for <i>st_blocks</i> in <sys/param.h> as the	1
12830	symbol DEV_BSIZE.	1
12831	<b>FUTURE DIRECTIONS</b>	
12832	No new S_IFMT symbolic names for the file type values of <b>mode_t</b> will be defined by	
12833	IEEE Std 1003.1-2001; if new file types are required, they will only be testable through <i>S_ISxx()</i>	
12834	or <i>S_TYPEISxxx()</i> macros instead.	
12835	<b>SEE ALSO</b>	
12836	<sys/statvfs.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, <i>chmod()</i> ,	1
12837	<i>fchmod()</i> , <i>fstat()</i> , <i>lstat()</i> , <i>mkdir()</i> , <i>mkfifo()</i> , <i>mknod()</i> , <i>stat()</i> , <i>umask()</i>	
12838	<b>CHANGE HISTORY</b>	
12839	First released in Issue 1. Derived from Issue 1 of the SVID.	
12840	<b>Issue 5</b>	
12841	The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.	
12842	The type of <i>st_blksize</i> is changed from <b>long</b> to <b>blksize_t</b> ; the type of <i>st_blocks</i> is changed from	
12843	<b>long</b> to <b>blkcnt_t</b> .	
12844	<b>Issue 6</b>	
12845	The <i>S_TYPEISMQ()</i> , <i>S_TYPEISSEM()</i> , and <i>S_TYPEISSHM()</i> macros are unconditionally	
12846	mandated.	
12847	The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types <b>blksize_t</b>	
12848	and <b>blkcnt_t</b> have been described.	
12849	The following new requirements on POSIX implementations derive from alignment with the	
12850	Single UNIX Specification:	
12851	• The <b>dev_t</b> , <b>ino_t</b> , <b>mode_t</b> , <b>nlink_t</b> , <b>uid_t</b> , <b>gid_t</b> , <b>off_t</b> , and <b>time_t</b> types are mandated.	
12852	<i>S_IFSOCK</i> and <i>S_ISSOCK</i> are added for sockets.	
12853	The description of <b>stat</b> structure members is changed to reflect contents when file type is a	
12854	symbolic link.	
12855	The test macro <i>S_TYPEISTMO</i> is added for alignment with IEEE Std 1003.1j-2000.	
12856	The <b>restrict</b> keyword is added to the prototypes for <i>lstat()</i> and <i>stat()</i> .	
12857	The <i>lstat()</i> function is made mandatory.	
12858	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the	1
12859	<i>st_blocks</i> member of the <b>stat</b> structure to the RATIONALE.	1

12860	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/25 is applied, adding to the	2
12861	DESCRIPTION that the <b>timespec</b> structure may be defined as described in the <time.h> header.	2

## 12862 NAME

12863 sys/statvfs.h — VFS File System information structure

## 12864 SYNOPSIS

12865 XSI `#include <sys/statvfs.h>`

12866

## 12867 DESCRIPTION

12868 The <sys/statvfs.h> header shall define the **statvfs** structure that includes at least the following  
12869 members:

12870	unsigned long	f_bsize	File system block size.
12871	unsigned long	f_frsize	Fundamental file system block size.
12872	fsblkcnt_t	f_blocks	Total number of blocks on file system in units of <i>f_frsize</i> .
12873	fsblkcnt_t	f_bfree	Total number of free blocks.
12874	fsblkcnt_t	f_bavail	Number of free blocks available to non-privileged process.
12875			
12876	fsfilcnt_t	f_files	Total number of file serial numbers.
12877	fsfilcnt_t	f_ffree	Total number of free file serial numbers.
12878	fsfilcnt_t	f_favail	Number of file serial numbers available to non-privileged process.
12879			
12880	unsigned long	f_fsid	File system ID.
12881	unsigned long	f_flag	Bit mask of <i>f_flag</i> values.
12882	unsigned long	f_namemax	Maximum filename length.

12883 The **fsblkcnt\_t** and **fsfilcnt\_t** types shall be defined as described in <sys/types.h>.

12884 The following flags for the *f\_flag* member shall be defined:

12885 **ST\_RDONLY** Read-only file system.

12886 **ST\_NOSUID** Does not support the semantics of the **ST\_ISUID** and **ST\_ISGID** file mode bits. 1

12887 The following shall be declared as functions and may also be defined as macros. Function  
12888 prototypes shall be provided.

12889 `int statvfs(const char *restrict, struct statvfs *restrict);`

12890 `int fstatvfs(int, struct statvfs *);`

## 12891 APPLICATION USAGE

12892 None.

## 12893 RATIONALE

12894 None.

## 12895 FUTURE DIRECTIONS

12896 None.

## 12897 SEE ALSO

12898 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *fstatvfs()*, *statvfs()*

## 12899 CHANGE HISTORY

12900 First released in Issue 4, Version 2.

## 12901 Issue 5

12902 The type of *f\_blocks*, *f\_bfree*, and *f\_bavail* is changed from **unsigned long** to **fsblkcnt\_t**; the type  
12903 of *f\_files*, *f\_ffree*, and *f\_favail* is changed from **unsigned long** to **fsfilcnt\_t**.

12904 **Issue 6**

12905 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt\_t**  
12906 and **fsfilcnt\_t** have been described.

12907 The **restrict** keyword is added to the prototype for *statvfs()*.

12908 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of 1  
12909 ST\_NOSUID from “Does not support *setuid()/setgid()* semantics” to “Does not support the 1  
12910 semantics of the ST\_ISUID and ST\_ISGID file mode bits”. 1

12911 **NAME**

12912 sys/time.h — time types

12913 **SYNOPSIS**

12914 XSI #include <sys/time.h>

12915

12916 **DESCRIPTION**

12917 The <sys/time.h> header shall define the **timeval** structure that includes at least the following  
12918 members:

12919	time_t	tv_sec	Seconds.
12920	suseconds_t	tv_usec	Microseconds.

12921 The <sys/time.h> header shall define the **itimerval** structure that includes at least the following  
12922 members:

12923	struct timeval it_interval	Timer interval.
12924	struct timeval it_value	Current value.

12925 The **time\_t** and **suseconds\_t** types shall be defined as described in <sys/types.h>.

12926 The **fd\_set** type shall be defined as described in <sys/select.h>.

12927 The <sys/time.h> header shall define the following values for the *which* argument of *getitimer()*  
12928 and *setitimer()*:

12929	ITIMER_REAL	Decrements in real time.
12930	ITIMER_VIRTUAL	Decrements in process virtual time.
12931	ITIMER_PROF	Decrements both in process virtual time and when the system is running
12932		on behalf of the process.

12933 The following shall be defined as described in <sys/select.h>:

12934 *FD\_CLR()*  
12935 *FD\_ISSET()*  
12936 *FD\_SET()*  
12937 *FD\_ZERO()*  
12938 *FD\_SETSIZE*

12939 The following shall be declared as functions and may also be defined as macros. Function  
12940 prototypes shall be provided.

```
12941 int getitimer(int, struct itimerval *);
12942 int gettimeofday(struct timeval *restrict, void *restrict);
12943 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12944           struct timeval *restrict);
12945 int setitimer(int, const struct itimerval *restrict,
12946             struct itimerval *restrict);
12947 int utimes(const char *, const struct timeval [2]); (LEGACY)
```

12948 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>  
12949 header.

12950 **APPLICATION USAGE**

12951       None.

12952 **RATIONALE**

12953       None.

12954 **FUTURE DIRECTIONS**

12955       None.

12956 **SEE ALSO**

12957       <sys/select.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *getitimer()*,  
12958       *gettimeofday()*, *select()*, *setitimer()*

12959 **CHANGE HISTORY**

12960       First released in Issue 4, Version 2.

12961 **Issue 5**12962       The type of *tv\_usec* is changed from **long** to **suseconds\_t**.12963 **Issue 6**12964       The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*.

12965       The note is added that inclusion of this header may also make symbols visible from  
12966       <sys/select.h>.

12967       The *utimes()* function is marked LEGACY.

12968 **NAME**

12969 sys/timeb.h — additional definitions for date and time

12970 **SYNOPSIS**

12971 XSI #include <sys/timeb.h>

12972

12973 **DESCRIPTION**

12974 The <sys/timeb.h> header shall define the **timeb** structure that includes at least the following  
12975 members:

12976	time_t	time	The seconds portion of the current time.
12977	unsigned short	millitm	The milliseconds portion of the current time.
12978	short	timezone	The local timezone in minutes west of Greenwich.
12979	short	dstflag	TRUE if Daylight Savings Time is in effect.

12980 The **time\_t** type shall be defined as described in <sys/types.h>.

12981 The following shall be declared as a function and may also be defined as a macro. A function  
12982 prototype shall be provided.

12983 int ftime(struct timeb \*); (**LEGACY**)

12984 **APPLICATION USAGE**

12985 None.

12986 **RATIONALE**

12987 None.

12988 **FUTURE DIRECTIONS**

12989 None.

12990 **SEE ALSO**

12991 <sys/types.h>, <time.h>

12992 **CHANGE HISTORY**

12993 First released in Issue 4, Version 2.

12994 **Issue 6**

12995 The *ftime()* function is marked LEGACY.

12996 **NAME**

12997       sys/times.h — file access and modification times structure

12998 **SYNOPSIS**

12999       #include &lt;sys/times.h&gt;

13000 **DESCRIPTION**13001       The <sys/times.h> header shall define the structure **tms**, which is returned by *times()* and  
13002       includes at least the following members:

13003       clock\_t   tms\_utime   User CPU time.

13004       clock\_t   tms\_stime   System CPU time.

13005       clock\_t   tms\_cutime   User CPU time of terminated child processes.

13006       clock\_t   tms\_cstime   System CPU time of terminated child processes.

13007       The **clock\_t** type shall be defined as described in <sys/types.h>.13008       The following shall be declared as a function and may also be defined as a macro. A function  
13009       prototype shall be provided.

13010       clock\_t times(struct tms \*);

13011 **APPLICATION USAGE**

13012       None.

13013 **RATIONALE**

13014       None.

13015 **FUTURE DIRECTIONS**

13016       None.

13017 **SEE ALSO**13018       <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *times()*13019 **CHANGE HISTORY**

13020       First released in Issue 1. Derived from Issue 1 of the SVID.



13021 **NAME**

13022 sys/types.h — data types

13023 **SYNOPSIS**

13024 #include <sys/types.h>

13025 **DESCRIPTION**

13026 The <sys/types.h> header shall include definitions for at least the following types:

13027	<b>blkcnt_t</b>	Used for file block counts.
13028	<b>blksize_t</b>	Used for block sizes.
13029 XSI 13030	<b>clock_t</b>	Used for system times in clock ticks or CLOCKS_PER_SEC; see <time.h>.
13031 TMR	<b>clockid_t</b>	Used for clock ID type in the clock and timer functions.
13032	<b>dev_t</b>	Used for device IDs.
13033 XSI	<b>fsblkcnt_t</b>	Used for file system block counts.
13034 XSI	<b>fsfilcnt_t</b>	Used for file system file counts.
13035	<b>gid_t</b>	Used for group IDs.
13036 XSI 13037	<b>id_t</b>	Used as a general identifier; can be used to contain at least a <b>pid_t</b> , <b>uid_t</b> , or <b>gid_t</b> .
13038	<b>ino_t</b>	Used for file serial numbers.
13039 XSI	<b>key_t</b>	Used for XSI interprocess communication.
13040	<b>mode_t</b>	Used for some file attributes.
13041	<b>nlink_t</b>	Used for link counts.
13042	<b>off_t</b>	Used for file sizes.
13043	<b>pid_t</b>	Used for process IDs and process group IDs.
13044 THR	<b>pthread_attr_t</b>	Used to identify a thread attribute object.
13045 BAR	<b>pthread_barrier_t</b>	Used to identify a barrier.
13046 BAR	<b>pthread_barrierattr_t</b>	Used to define a barrier attributes object.
13047 THR	<b>pthread_cond_t</b>	Used for condition variables.
13048 THR	<b>pthread_condattr_t</b>	Used to identify a condition attribute object.
13049 THR	<b>pthread_key_t</b>	Used for thread-specific data keys.
13050 THR	<b>pthread_mutex_t</b>	Used for mutexes.
13051 THR	<b>pthread_mutexattr_t</b>	Used to identify a mutex attribute object.
13052 THR	<b>pthread_once_t</b>	Used for dynamic package initialization.
13053 THR	<b>pthread_rwlock_t</b>	Used for read-write locks.
13054 THR	<b>pthread_rwlockattr_t</b>	Used for read-write lock attributes.
13055 SPI	<b>pthread_spinlock_t</b>	Used to identify a spin lock.
13056 THR	<b>pthread_t</b>	Used to identify a thread.

13057		<b>size_t</b>	Used for sizes of objects.
13058		<b>ssize_t</b>	Used for a count of bytes or an error indication.
13059	XSI	<b>suseconds_t</b>	Used for time in microseconds.
13060		<b>time_t</b>	Used for time in seconds.
13061	TMR	<b>timer_t</b>	Used for timer ID returned by <i>timer_create()</i> .
13062	TRC	<b>trace_attr_t</b>	Used to identify a trace stream attributes object.
13063	TRC	<b>trace_event_id_t</b>	Used to identify a trace event type.
13064	TRC TEF	<b>trace_event_set_t</b>	Used to identify a trace event type set.
13065	TRC	<b>trace_id_t</b>	Used to identify a trace stream.
13066		<b>uid_t</b>	Used for user IDs.
13067	XSI	<b>useconds_t</b>	Used for time in microseconds.
13068		All of the types shall be defined as arithmetic types of an appropriate length, with the following exceptions:	
13070	XSI	<b>key_t</b>	
13071	THR	<b>pthread_attr_t</b>	
13072	BAR	<b>pthread_barrier_t</b>	
13073		<b>pthread_barrierattr_t</b>	
13074	THR	<b>pthread_cond_t</b>	
13075		<b>pthread_condattr_t</b>	
13076		<b>pthread_key_t</b>	
13077		<b>pthread_mutex_t</b>	
13078		<b>pthread_mutexattr_t</b>	
13079		<b>pthread_once_t</b>	
13080		<b>pthread_rwlock_t</b>	
13081		<b>pthread_rwlockattr_t</b>	
13082	SPI	<b>pthread_spinlock_t</b>	
13083	THR	<b>pthread_t</b>	
13084	TRC	<b>trace_attr_t</b>	
13085		<b>trace_event_id_t</b>	
13086	TRC TEF	<b>trace_event_set_t</b>	
13087	TRC	<b>trace_id_t</b>	
13088			
13089		Additionally:	
13090		• <b>mode_t</b> shall be an integer type.	
13091		• <b>nlink_t</b> , <b>uid_t</b> , <b>gid_t</b> , and <b>id_t</b> shall be integer types.	
13092		• <b>blkcnt_t</b> and <b>off_t</b> shall be signed integer types.	
13093	XSI	• <b>fsblkcnt_t</b> , <b>fsfilcnt_t</b> , and <b>ino_t</b> shall be defined as unsigned integer types.	
13094		• <b>size_t</b> shall be an unsigned integer type.	
13095		• <b>blksize_t</b> , <b>pid_t</b> , and <b>ssize_t</b> shall be signed integer types.	
13096		• <b>time_t</b> and <b>clock_t</b> shall be integer or real-floating types.	
13097	XSI	The type <b>ssize_t</b> shall be capable of storing values at least in the range $[-1, \{\text{SSIZE\_MAX}\}]$ . The	
13098		type <b>useconds_t</b> shall be an unsigned integer type capable of storing values at least in the range	

13099 [0, 1 000 000]. The type **suseconds\_t** shall be a signed integer type capable of storing values at  
13100 least in the range [-1, 1 000 000].

13101 The implementation shall support one or more programming environments in which the widths  
13102 of **blksize\_t**, **pid\_t**, **size\_t**, **ssize\_t**, **suseconds\_t**, and **useconds\_t** are no greater than the width of  
13103 type **long**. The names of these programming environments can be obtained using the *confstr()*  
13104 function or the *getconf* utility.

13105 There are no defined comparison or assignment operators for the following types:

13106	THR	<b>pthread_attr_t</b>
13107	BAR	<b>pthread_barrier_t</b>
13108		<b>pthread_barrierattr_t</b>
13109	THR	<b>pthread_cond_t</b>
13110		<b>pthread_condattr_t</b>
13111		<b>pthread_mutex_t</b>
13112		<b>pthread_mutexattr_t</b>
13113		<b>pthread_rwlock_t</b>
13114		<b>pthread_rwlockattr_t</b>
13115	SPI	<b>pthread_spinlock_t</b>
13116	TRC	<b>trace_attr_t</b>
13117		

## 13118 APPLICATION USAGE

13119 None.

## 13120 RATIONALE

13121 None.

## 13122 FUTURE DIRECTIONS

13123 None.

## 13124 SEE ALSO

13125 <time.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *confstr()*, the Shell and Utilities  
13126 volume of IEEE Std 1003.1-2001, *getconf*

## 13127 CHANGE HISTORY

13128 First released in Issue 1. Derived from Issue 1 of the SVID.

### 13129 Issue 5

13130 The **clockid\_t** and **timer\_t** types are defined for alignment with the POSIX Realtime Extension.

13131 The types **blkcnt\_t**, **blksize\_t**, **fsblkcnt\_t**, **fsfilcnt\_t**, and **suseconds\_t** are added.

13132 Large File System extensions are added.

13133 Updated for alignment with the POSIX Threads Extension.

### 13134 Issue 6

13135 The **pthread\_barrier\_t**, **pthread\_barrierattr\_t**, and **pthread\_spinlock\_t** types are added for  
13136 alignment with IEEE Std 1003.1j-2000.

13137 The margin code is changed from XSI to THR for the **pthread\_rwlock\_t** and  
13138 **pthread\_rwlockattr\_t** types as Read-Write Locks have been absorbed into the POSIX Threads  
13139 option. The threads types are marked THR.

13140 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/26 is applied, adding **pthread\_t** to the list 2  
13141 of types that are not required to be arithmetic types, thus allowing **pthread\_t** to be defined as a 2  
13142 structure. 2

13143 **NAME**

13144 sys/uio.h — definitions for vector I/O operations

13145 **SYNOPSIS**13146 XSI `#include <sys/uio.h>`

13147

13148 **DESCRIPTION**13149 The <sys/uio.h> header shall define the **iovec** structure that includes at least the following  
13150 members:13151 `void *iov_base` Base address of a memory region for input or output.13152 `size_t iov_len` The size of the memory pointed to by *iov\_base*.13153 The <sys/uio.h> header uses the **iovec** structure for scatter/gather I/O.13154 The **ssize\_t** and **size\_t** types shall be defined as described in <sys/types.h>.13155 The following shall be declared as functions and may also be defined as macros. Function  
13156 prototypes shall be provided.13157 `ssize_t readv(int, const struct iovec *, int);`13158 `ssize_t writev(int, const struct iovec *, int);`13159 **APPLICATION USAGE**13160 The implementation can put a limit on the number of scatter/gather elements which can be  
13161 processed in one call. The symbol {IOV\_MAX} defined in <limits.h> should always be used to  
13162 learn about the limits instead of assuming a fixed value.13163 **RATIONALE**13164 Traditionally, the maximum number of scatter/gather elements the system can process in one  
13165 call were described by the symbolic value {UIO\_MAXIOV}. In IEEE Std 1003.1-2001 this value is  
13166 replaced by the constant {IOV\_MAX} which can be found in <limits.h>.13167 **FUTURE DIRECTIONS**

13168 None.

13169 **SEE ALSO**13170 <limits.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *read()*, *write()*13171 **CHANGE HISTORY**

13172 First released in Issue 4, Version 2.

13173 **Issue 6**

13174 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13175 **NAME**

13176       sys/un.h — definitions for UNIX domain sockets

13177 **SYNOPSIS**

13178       #include <sys/un.h>

13179 **DESCRIPTION**

13180       The <sys/un.h> header shall define the **sockaddr\_un** structure that includes at least the  
13181       following members:

13182       sa\_family\_t   sun\_family   Address family.  
13183       char           sun\_path[]   Socket pathname.

13184       The **sockaddr\_un** structure is used to store addresses for UNIX domain sockets. Values of this  
13185       type shall be cast by applications to **struct sockaddr** for use with socket functions.

13186       The **sa\_family\_t** type shall be defined as described in <sys/socket.h>.

13187 **APPLICATION USAGE**

13188       The size of *sun\_path* has intentionally been left undefined. This is because different  
13189       implementations use different sizes. For example, 4.3 BSD uses a size of 108, and 4.4 BSD uses a  
13190       size of 104. Since most implementations originate from BSD versions, the size is typically in the  
13191       range 92 to 108.

13192       Applications should not assume a particular length for *sun\_path* or assume that it can hold  
13193       {\_POSIX\_PATH\_MAX} characters (255).

13194 **RATIONALE**

13195       None.

13196 **FUTURE DIRECTIONS**

13197       None.

13198 **SEE ALSO**

13199       <sys/socket.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *bind()*, *socket()*,  
13200       *socketpair()*

13201 **CHANGE HISTORY**

13202       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13203 **NAME**

13204 sys/utsname.h — system name structure

13205 **SYNOPSIS**

13206 #include &lt;sys/utsname.h&gt;

13207 **DESCRIPTION**

13208 The <sys/utsname.h> header shall define the structure **utsname** which shall include at least the  
 13209 following members:

13210	char	sysname[ ]	Name of this implementation of the operating system.	
13211	char	nodename[ ]	Name of this node within the communications	2
13212			network to which this node is attached, if any.	2
13213	char	release[ ]	Current release level of this implementation.	
13214	char	version[ ]	Current version level of this release.	
13215	char	machine[ ]	Name of the hardware type on which the system is running.	

13216 The character arrays are of unspecified size, but the data stored in them shall be terminated by a  
 13217 null byte.

13218 The following shall be declared as a function and may also be defined as a macro:

13219 int uname(struct utsname \*);

13220 **APPLICATION USAGE**

13221 None.

13222 **RATIONALE**

13223 None.

13224 **FUTURE DIRECTIONS**

13225 None.

13226 **SEE ALSO**13227 The System Interfaces volume of IEEE Std 1003.1-2001, *uname()*13228 **CHANGE HISTORY**

13229 First released in Issue 1. Derived from Issue 1 of the SVID.

13230 **Issue 6**

13231	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/27 is applied, changing the description of	2
13232	<i>nodename</i> within the <b>utsname</b> structure from “an implementation-defined communications	2
13233	network” to “the communications network to which this node is attached, if any”.	2

13234 **NAME**

13235 sys/wait.h — declarations for waiting

13236 **SYNOPSIS**

13237 #include <sys/wait.h>

13238 **DESCRIPTION**

13239 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

13240 WNOHANG Do not hang if no status is available; return immediately.

13241 WUNTRACED Report status of stopped child process.

13242 The <sys/wait.h> header shall define the following macros for analysis of process status values:

13243 WEXITSTATUS Return exit status.

13244 XSI WIFCONTINUED True if child has been continued.

13245 WIFEXITED True if child exited normally.

13246 WIFSIGNALED True if child exited due to uncaught signal.

13247 WIFSTOPPED True if child is currently stopped.

13248 WSTOPSIG Return signal number that caused process to stop.

13249 WTERMSIG Return signal number that caused process to terminate.

13250 XSI The following symbolic constants shall be defined as possible values for the *options* argument to  
13251 *waitid()*:

13252 WEXITED Wait for processes that have exited.

13253 WSTOPPED Status is returned for any child that has stopped upon receipt of a signal.

13254 WCONTINUED Status is returned for any child that was stopped and has been continued.

13255 WNOHANG Return immediately if there are no children to wait for.

13256 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state.

13257 The type **idtype\_t** shall be defined as an enumeration type whose possible values shall include  
13258 at least the following:

13259 P\_ALL

13260 P\_PID

13261 P\_PGID

13262

13263 The **id\_t** and **pid\_t** types shall be defined as described in <sys/types.h>.

13264 XSI The **siginfo\_t** type shall be defined as described in <signal.h>.

13265 The **rusage** structure shall be defined as described in <sys/resource.h>.

13266 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h> and  
13267 <sys/resource.h>.

13268 The following shall be declared as functions and may also be defined as macros. Function  
13269 prototypes shall be provided.

13270 pid\_t wait(int \*);

13271 XSI int waitid(idtype\_t, id\_t, siginfo\_t \*, int);

13272 pid\_t waitpid(pid\_t, int \*, int);

13273 **APPLICATION USAGE**

13274       None.

13275 **RATIONALE**

13276       None.

13277 **FUTURE DIRECTIONS**

13278       None.

13279 **SEE ALSO**

13280       &lt;signal.h&gt;, &lt;sys/resource.h&gt;, &lt;sys/types.h&gt;, the System Interfaces volume of

13281       IEEE Std 1003.1-2001, *wait()*, *waitid()*13282 **CHANGE HISTORY**

13283       First released in Issue 3.

13284       Included for alignment with the POSIX.1-1988 standard.

13285 **Issue 6**13286       The *wait3()* function is removed.



13287 **NAME**

13288       syslog.h — definitions for system error logging

13289 **SYNOPSIS**

13290 xSI       #include <syslog.h>

13291

13292 **DESCRIPTION**

13293       The <syslog.h> header shall define the following symbolic constants, zero or more of which may  
13294       be OR'ed together to form the *logopt* option of *openlog()*:

13295       LOG\_PID               Log the process ID with each message.

13296       LOG\_CONS             Log to the system console on error.

13297       LOG\_NDELAY           Connect to syslog daemon immediately.

13298       LOG\_ODELAY           Delay open until *syslog()* is called.

13299       LOG\_NOWAIT           Do not wait for child processes.

13300       The following symbolic constants shall be defined as possible values of the *facility* argument to  
13301       *openlog()*:

13302       LOG\_KERN             Reserved for message generated by the system.

13303       LOG\_USER             Message generated by a process.

13304       LOG\_MAIL             Reserved for message generated by mail system.

13305       LOG\_NEWS             Reserved for message generated by news system.

13306       LOG\_UUCP             Reserved for message generated by UUCP system.

13307       LOG\_DAEMON           Reserved for message generated by system daemon.

13308       LOG\_AUTH             Reserved for message generated by authorization daemon.

13309       LOG\_CRON             Reserved for message generated by clock daemon.

13310       LOG\_LPR              Reserved for message generated by printer system.

13311       LOG\_LOCAL0           Reserved for local use.

13312       LOG\_LOCAL1           Reserved for local use.

13313       LOG\_LOCAL2           Reserved for local use.

13314       LOG\_LOCAL3           Reserved for local use.

13315       LOG\_LOCAL4           Reserved for local use.

13316       LOG\_LOCAL5           Reserved for local use.

13317       LOG\_LOCAL6           Reserved for local use.

13318       LOG\_LOCAL7           Reserved for local use.

13319       The following shall be declared as macros for constructing the *maskpri* argument to *setlogmask()*.  
13320       The following macros expand to an expression of type **int** when the argument *pri* is an  
13321       expression of type **int**:

13322       LOG\_MASK(*pri*)       A mask for priority *pri*.

13323       The following constants shall be defined as possible values for the *priority* argument of *syslog()*:

13324	LOG_EMERG	A panic condition was reported to all processes.
13325	LOG_ALERT	A condition that should be corrected immediately.
13326	LOG_CRIT	A critical condition.
13327	LOG_ERR	An error message.
13328	LOG_WARNING	A warning message.
13329	LOG_NOTICE	A condition requiring special handling.
13330	LOG_INFO	A general information message.
13331	LOG_DEBUG	A message useful for debugging programs.
13332	The following shall be declared as functions and may also be defined as macros. Function	
13333	prototypes shall be provided.	
13334	void	closelog(void);
13335	void	openlog(const char *, int, int);
13336	int	setlogmask(int);
13337	void	syslog(int, const char *, ...);
13338	<b>APPLICATION USAGE</b>	
13339	None.	
13340	<b>RATIONALE</b>	
13341	None.	
13342	<b>FUTURE DIRECTIONS</b>	
13343	None.	
13344	<b>SEE ALSO</b>	
13345	The System Interfaces volume of IEEE Std 1003.1-2001, <i>closelog()</i>	
13346	<b>CHANGE HISTORY</b>	
13347	First released in Issue 4, Version 2.	
13348	<b>Issue 5</b>	
13349	Moved from X/Open UNIX to BASE.	

13350 **NAME**

13351 tar.h — extended tar definitions

13352 **SYNOPSIS**

13353 #include <tar.h>

13354 **DESCRIPTION**

13355 The <tar.h> header shall define header block definitions as follows.

13356 General definitions:

13357

13358

13359

13360

13361

13362

Name	Description	Value
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLEN	2	Length of the above.

13363

*Typeflag* field definitions:

13364

13365

13366

13367

13368

13369

13370

13371

13372

13373

13374

Name	Description	Value
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

13375

*Mode* field bit definitions (octal):

13376

13377

13378

13379

13380 XSI

13381

13382

13383

13384

13385

13386

13387

13388

13389

Name	Description	Value
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

13390 **APPLICATION USAGE**

13391           None.

13392 **RATIONALE**

13393           None.

13394 **FUTURE DIRECTIONS**

13395           None.

13396 **SEE ALSO**13397           The Shell and Utilities volume of IEEE Std 1003.1-2001, *pax*13398 **CHANGE HISTORY**

13399           First released in Issue 3. Derived from the POSIX.1-1988 standard.

13400 **Issue 6**13401           The SEE ALSO section now refers to *pax* since the Shell and Utilities volume of  
13402           IEEE Std 1003.1-2001 no longer contains the *tar* utility.

13403 **NAME**

13404 termios.h — define values for termios

13405 **SYNOPSIS**

13406 #include <termios.h>

13407 **DESCRIPTION**

13408 The <termios.h> header contains the definitions used by the terminal I/O interfaces (see  
13409 Chapter 11 (on page 187) for the structures and names defined).

13410 **The termios Structure**

13411 The following data types shall be defined through **typedef**:

13412 **cc\_t** Used for terminal special characters.

13413 **speed\_t** Used for terminal baud rates.

13414 **tcflag\_t** Used for terminal modes.

13415 The above types shall be all unsigned integer types.

13416 The implementation shall support one or more programming environments in which the widths  
13417 of **cc\_t**, **speed\_t**, and **tcflag\_t** are no greater than the width of type **long**. The names of these  
13418 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

13419 The **termios** structure shall be defined, and shall include at least the following members:

13420 tcflag\_t c\_iflag Input modes.  
13421 tcflag\_t c\_oflag Output modes.  
13422 tcflag\_t c\_cflag Control modes.  
13423 tcflag\_t c\_lflag Local modes.  
13424 cc\_t c\_cc[NCCS] Control characters.

13425 A definition shall be provided for:

13426 NCCS Size of the array *c\_cc* for control characters.

13427 The following subscript names for the array *c\_cc* shall be defined:

13428

13429

13430

13431

13432

13433

13434

13435

13436

13437

13438

13439

13440

13441

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF	VINTR	EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR		INTR character.
VKILL	VMIN	KILL character.
		MIN value.
VQUIT		QUIT character.
VSTART		START character.
VSTOP	VSUSP	STOP character.
VSUSP		SUSP character.
	VTIME	TIME value.

13442 The subscript values shall be unique, except that the VMIN and VTIME subscripts may have the  
13443 same values as the VEOF and VEOL subscripts, respectively.

13444 The following flags shall be provided.

13445 **Input Modes**13446 The *c\_iflag* field describes the basic terminal input control:

13447	BRKINT	Signal interrupt on break.
13448	ICRNL	Map CR to NL on input.
13449	IGNBRK	Ignore break condition.
13450	IGNCR	Ignore CR.
13451	IGNPAR	Ignore characters with parity errors.
13452	INLCR	Map NL to CR on input.
13453	INPCK	Enable input parity check.
13454	ISTRIP	Strip character.
13455 XSI	IXANY	Enable any character to restart output.
13456	IXOFF	Enable start/stop input control.
13457	IXON	Enable start/stop output control.
13458	PARMRK	Mark parity errors.

13459 **Output Modes**13460 The *c\_oflag* field specifies the system treatment of output:

13461	OPOST	Post-process output.
13462 XSI	ONLCR	Map NL to CR-NL on output.
13463	OCRNL	Map CR to NL on output.
13464	ONOCR	No CR output at column 0.
13465	ONLRET	NL performs CR function.
13466	OFILL	Use fill characters for delay.
13467	NLDLY	Select newline delays:
13468	NL0	Newline type 0.
13469	NL1	Newline type 1.
13470	CRDLY	Select carriage-return delays:
13471	CR0	Carriage-return delay type 0.
13472	CR1	Carriage-return delay type 1.
13473	CR2	Carriage-return delay type 2.
13474	CR3	Carriage-return delay type 3.
13475	TABDLY	Select horizontal-tab delays:
13476	TAB0	Horizontal-tab delay type 0.
13477	TAB1	Horizontal-tab delay type 1.
13478	TAB2	Horizontal-tab delay type 2.

13479	TAB3	Expand tabs to spaces.
13480	BSDLY	Select backspace delays:
13481	BS0	Backspace-delay type 0.
13482	BS1	Backspace-delay type 1.
13483	VTDLY	Select vertical-tab delays:
13484	VT0	Vertical-tab delay type 0.
13485	VT1	Vertical-tab delay type 1.
13486	FFDLY	Select form-feed delays:
13487	FF0	Form-feed delay type 0.
13488	FF1	Form-feed delay type 1.

13489 **Baud Rate Selection**

13490 The input and output baud rates are stored in the **termios** structure. These are the valid values  
 13491 for objects of type **speed\_t**. The following values shall be defined, but not all baud rates need be  
 13492 supported by the underlying hardware.

13493	B0	Hang up
13494	B50	50 baud
13495	B75	75 baud
13496	B110	110 baud
13497	B134	134.5 baud
13498	B150	150 baud
13499	B200	200 baud
13500	B300	300 baud
13501	B600	600 baud
13502	B1200	1 200 baud
13503	B1800	1 800 baud
13504	B2400	2 400 baud
13505	B4800	4 800 baud
13506	B9600	9 600 baud
13507	B19200	19 200 baud
13508	B38400	38 400 baud

13509      **Control Modes**

13510      The *c\_cflag* field describes the hardware control of the terminal; not all values specified are  
 13511      required to be supported by the underlying hardware:

13512      CSIZE            Character size:

13513                   CS5        5 bits

13514                   CS6        6 bits

13515                   CS7        7 bits

13516                   CS8        8 bits

13517      CSTOPB        Send two stop bits, else one.

13518      CREAD        Enable receiver.

13519      PARENB        Parity enable.

13520      PARODD        Odd parity, else even.

13521      HUPCL        Hang up on last close.

13522      CLOCAL        Ignore modem status lines.

13523      The implementation shall support the functionality associated with the symbols CS7, CS8,  
 13524      CSTOPB, PARODD, and PARENB.

13525      **Local Modes**

13526      The *c\_lflag* field of the argument structure is used to control various terminal functions:

13527      ECHO            Enable echo.

13528      ECHOE            Echo erase character as error-correcting backspace.

13529      ECHOK            Echo KILL.

13530      ECHONL          Echo NL.

13531      ICANON          Canonical input (erase and kill processing).

13532      IEXTEN          Enable extended input character processing.

13533      ISIG            Enable signals.

13534      NOFLSH          Disable flush after interrupt or quit.

13535      TOSTOP          Send SIGTTOU for background output.

13536      **Attribute Selection**

13537      The following symbolic constants for use with *tcsetattr()* are defined:

13538      TCSANOW          Change attributes immediately.

13539      TCSADRAIN        Change attributes when output has drained.

13540      TCSAFLUSH        Change attributes when output has drained; also flush pending input.



13541 **Line Control**

13542 The following symbolic constants for use with *tcflush()* shall be defined:

13543 TCIFLUSH Flush pending input.

13544 TCIOFLUSH Flush both pending input and untransmitted output.

13545 TCOFLUSH Flush untransmitted output.

13546 The following symbolic constants for use with *tcflow()* shall be defined:

13547 TCIOFF Transmit a STOP character, intended to suspend input data.

13548 TCION Transmit a START character, intended to restart input data.

13549 TCOOFF Suspend output.

13550 TCOON Restart output.

13551 The following shall be declared as functions and may also be defined as macros. Function  
13552 prototypes shall be provided.

```
13553 speed_t cfgetispeed(const struct termios *);
13554 speed_t cfgetospeed(const struct termios *);
13555 int cfsetispeed(struct termios *, speed_t);
13556 int cfsetospeed(struct termios *, speed_t);
13557 int tcdrain(int);
13558 int tcflow(int, int);
13559 int tcflush(int, int);
13560 int tcgetattr(int, struct termios *);
13561 XSI pid_t tcgetsid(int);
13562 int tcsendbreak(int, int);
13563 int tcsetattr(int, int, const struct termios *);
```

13564 **APPLICATION USAGE**

13565 The following names are reserved for XSI-conformant systems to use as an extension to the  
13566 above; therefore strictly conforming applications shall not use them:

13567	CBAUD	EXTB	VDSUSP
13568	DEFECHO	FLUSHO	VLNEXT
13569	ECHOCTL	LOBLK	VREPRINT
13570	ECHOKE	PENDIN	VSTATUS
13571	ECHOPRT	SWTCH	VWERASE
13572	EXTA	VDISCARD	

13573 **RATIONALE**

13574 None.

13575 **FUTURE DIRECTIONS**

13576 None.

13577 **SEE ALSO**

13578 The System Interfaces volume of IEEE Std 1003.1-2001, *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*,  
13579 *cfsetospeed()*, *confstr()*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()*,  
13580 the Shell and Utilities volume of IEEE Std 1003.1-2001, *getconf*, Chapter 11 (on page 187)

13581 **CHANGE HISTORY**

13582 First released in Issue 3.

13583 Included for alignment with the ISO POSIX-1 standard.

13584 **Issue 6**

13585 The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.

13586 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are  
13587 reaffirmed.13588 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to 1  
13589 ECHOKE in the APPLICATION USAGE section. 1

13590 **NAME**

13591 tgmath.h — type-generic macros

13592 **SYNOPSIS**

13593 #include <tgmath.h>

13594 **DESCRIPTION**

13595 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
13596 conflict between the requirements described here and the ISO C standard is unintentional. This  
13597 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13598 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define  
13599 several type-generic macros.

13600 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**) or  
13601 *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is  
13602 **double**. For each such function, except *modf()*, there shall be a corresponding type-generic  
13603 macro. The parameters whose corresponding real type is **double** in the function synopsis are  
13604 generic parameters. Use of the macro invokes a function whose corresponding real type and  
13605 type domain are determined by the arguments for the generic parameters.

13606 Use of the macro invokes a function whose generic parameters have the corresponding real type  
13607 determined as follows:

- 13608 • First, if any argument for generic parameters has type **long double**, the type determined is  
13609 **long double**.
- 13610 • Otherwise, if any argument for generic parameters has type **double** or is of integer type, the  
13611 type determined is **double**.
- 13612 • Otherwise, the type determined is **float**.

13613 For each unsuffixed function in the <math.h> header for which there is a function in the  
13614 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic  
13615 macro (for both functions) has the same name as the function in the <math.h> header. The  
13616 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

13617	<math.h>	<complex.h>	Type-Generic
13618	Function	Function	Macro
13619	<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
13620	<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
13621	<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
13622	<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
13623	<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
13624	<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
13625	<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
13626	<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
13627	<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
13628	<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
13629	<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
13630	<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
13631	<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
13632	<i>log()</i>	<i>clog()</i>	<i>log()</i>
13633	<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
13634	<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
13635	<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

13636 If at least one argument for a generic parameter is complex, then use of the macro invokes a  
 13637 complex function; otherwise, use of the macro invokes a real function.

13638 For each unsuffixed function in the <math.h> header without a *c*-prefixed counterpart in the  
 13639 <complex.h> header, the corresponding type-generic macro has the same name as the function.  
 13640 These type-generic macros are:

13641	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
13642	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
13643	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
13644	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
13645	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>
13646	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>
13647	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
13648	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
13649	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
13650	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

13651 If all arguments for generic parameters are real, then use of the macro invokes a real function;  
 13652 otherwise, use of the macro results in undefined behavior.

13653 For each unsuffixed function in the <complex.h> header that is not a *c*-prefixed counterpart to a  
 13654 function in the <math.h> header, the corresponding type-generic macro has the same name as  
 13655 the function. These type-generic macros are:

13656 *carg()*  
 13657 *cimag()*  
 13658 *conj()*  
 13659 *cproj()*  
 13660 *creal()*

13661 Use of the macro with any real or complex argument invokes a complex function.

## 13662 APPLICATION USAGE

13663 With the declarations:

```
13664 #include <tgmath.h>
13665 int n;
13666 float f;
13667 double d;
13668 long double ld;
13669 float complex fc;
13670 double complex dc;
13671 long double complex ldc;
```

13672 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acoshf(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>

13679

13680

13681

13682

13683

13684

13685

13686

13687

13688

13689

13690

13691

13692

13693

13694

13695

13696

13697

Macro	Use Invokes
<i>log(fc)</i>	<i>clogf(fc)</i>
<i>sqr(d)</i>	<i>csqr(d)</i>
<i>pow(l,d,f)</i>	<i>cpowl(l,d,f)</i>
<i>remainder(n,n)</i>	<i>remainder(n,n)</i> , the function
<i>nextafter(d,f)</i>	<i>nextafter(d,f)</i> , the function
<i>nexttoward(f,l)</i>	<i>nexttowardf(f,l)</i>
<i>copysign(n,l)</i>	<i>copysignl(n,l)</i>
<i>ceil(fc)</i>	Undefined behavior
<i>rint(d)</i>	Undefined behavior
<i>fmax(l,d,l)</i>	Undefined behavior
<i>carg(n)</i>	<i>carg(n)</i> , the function
<i>cproj(f)</i>	<i>cprojf(f)</i>
<i>creal(d)</i>	<i>creal(d)</i> , the function
<i>cimag(l)</i>	<i>cimagl(l)</i>
<i>cabs(fc)</i>	<i>cabsf(fc)</i>
<i>carg(d)</i>	<i>carg(d)</i> , the function
<i>cproj(l)</i>	<i>cprojl(l)</i>

## 13698 RATIONALE

13699

13700

13701

13702

13703

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '\*'. For example, with a type-generic *cos()* macro, the expression *cos(float)x* will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

13704

13705

13706

13707

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

13708

13709

13710

The term "type-generic" was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

13711

13712

The macros are placed in their own header in order not to silently break old programs that include the <math.h> header; for example, with:

13713

```
printf ("%e", sin(x))
```

13714

13715

*modf(double, double \*)* is excluded because no way was seen to make it safe without complicating the type resolution.

13716

13717

13718

The implementation might, as an extension, endow appropriate ones of the macros that IEEE Std 1003.1-2001 specifies only for real arguments with the ability to invoke the complex functions.

13719

13720

13721

IEEE Std 1003.1-2001 does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqr()*, for example, could be implemented with:

13722

13723

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

13724

13725

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

13726 The great majority of existing C programs are expected to be unaffected when the <tgmath.h>  
13727 header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar  
13728 to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return  
13729 values differ.

13730 The ability to overload on integer as well as floating types would have been useful for some  
13731 functions; for example, *copysign()*. Overloading with different numbers of arguments would  
13732 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities  
13733 would have complicated the specification; and their natural consistent use, such as for a floating  
13734 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the  
13735 ISO/IEC 9899:1999 standard for insufficient benefit.

13736 The ISO C standard in no way limits the implementation's options for efficiency, including  
13737 inlining library functions.

#### 13738 FUTURE DIRECTIONS

13739 None.

#### 13740 SEE ALSO

13741 <math.h>, <complex.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *cabs()*, *fabs()*,  
13742 *modf()*

#### 13743 CHANGE HISTORY

13744 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

13745 **NAME**

13746       time.h — time types

13747 **SYNOPSIS**

13748       #include <time.h>

13749 **DESCRIPTION**

13750 CX       Some of the functionality described on this reference page extends the ISO C standard.  
13751       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
13752       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
13753       symbols in this header.

13754       The <time.h> header shall declare the structure **tm**, which shall include at least the following  
13755       members:

13756	int	tm_sec	Seconds [0,60].
13757	int	tm_min	Minutes [0,59].
13758	int	tm_hour	Hour [0,23].
13759	int	tm_mday	Day of month [1,31].
13760	int	tm_mon	Month of year [0,11].
13761	int	tm_year	Years since 1900.
13762	int	tm_wday	Day of week [0,6] (Sunday =0).
13763	int	tm_yday	Day of year [0,365].
13764	int	tm_isdst	Daylight Savings flag.

13765       The value of *tm\_isdst* shall be positive if Daylight Savings Time is in effect, 0 if Daylight Savings  
13766       Time is not in effect, and negative if the information is not available.

13767       The <time.h> header shall define the following symbolic names:

13768	NULL	Null pointer constant.
13769	CLOCKS_PER_SEC	A number used to convert the value returned by the <i>clock()</i> function into 13770       seconds.

13771	TMR CPT	<b>CLOCK_PROCESS_CPUTIME_ID</b>
13772		The identifier of the CPU-time clock associated with the process making a 13773 <i>clock()</i> or <i>timer*()</i> function call.

13774	TMR TCT	<b>CLOCK_THREAD_CPUTIME_ID</b>
13775		The identifier of the CPU-time clock associated with the thread making a 13776 <i>clock()</i> or <i>timer*()</i> function call.

13777 TMR       The <time.h> header shall declare the structure **timespec**, which has at least the following  
13778       members:

13779	time_t	tv_sec	Seconds.
13780	long	tv_nsec	Nanoseconds.

13781       The <time.h> header shall also declare the **itimerspec** structure, which has at least the following  
13782       members:

13783	struct timespec	it_interval	Timer period.
13784	struct timespec	it_value	Timer expiration.

13785       The following manifest constants shall be defined:

13786	<b>CLOCK_REALTIME</b>	The identifier of the system-wide realtime clock.
13787	<b>TIMER_ABSTIME</b>	Flag indicating time is absolute. For functions taking timer objects, this 13788       refers to the clock associated with the timer.

13789 MON	CLOCK_MONOTONIC
13790	The identifier for the system-wide monotonic clock, which is defined as a
13791	clock whose value cannot be set via <i>clock_settime()</i> and which cannot
13792	have backward clock jumps. The maximum possible clock jump shall be
13793	implementation-defined.
13794 TMR	The <b>clock_t</b> , <b>size_t</b> , <b>time_t</b> , <b>clockid_t</b> , and <b>timer_t</b> types shall be defined as described in
13795	<sys/types.h>.
13796 XSI	Although the value of CLOCKS_PER_SEC is required to be 1 million on all XSI-conformant
13797	systems, it may be variable on other systems, and it should not be assumed that
13798	CLOCKS_PER_SEC is a compile-time constant.
13799 XSI	The <time.h> header shall provide a declaration for <i>getdate_err</i> .
13800	The following shall be declared as functions and may also be defined as macros. Function
13801	prototypes shall be provided.
13802	char       *asctime(const struct tm *);
13803 TSF	char       *asctime_r(const struct tm *restrict, char *restrict);
13804	clock_t     clock(void);
13805 CPT	int        clock_getcpuclockid(pid_t, clockid_t *);
13806 TMR	int        clock_getres(clockid_t, struct timespec *);
13807	int        clock_gettime(clockid_t, struct timespec *);
13808 CS	int        clock_nanosleep(clockid_t, int, const struct timespec *,
13809	struct timespec *);
13810 TMR	int        clock_settime(clockid_t, const struct timespec *);
13811	char       *ctime(const time_t *);
13812 TSF	char       *ctime_r(const time_t *, char *);
13813	double     difftime(time_t, time_t);
13814 XSI	struct tm *getdate(const char *);
13815	struct tm *gmtime(const time_t *);
13816 TSF	struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
13817	struct tm *localtime(const time_t *);
13818 TSF	struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
13819	time_t     mktime(struct tm *);
13820 TMR	int        nanosleep(const struct timespec *, struct timespec *);
13821	size_t     strftime(char *restrict, size_t, const char *restrict,
13822	const struct tm *restrict);
13823 XSI	char       *strptime(const char *restrict, const char *restrict,
13824	struct tm *restrict);
13825	time_t     time(time_t *);
13826 TMR	int        timer_create(clockid_t, struct sigevent *restrict,
13827	timer_t *restrict);
13828	int        timer_delete(timer_t);
13829	int        timer_gettime(timer_t, struct itimerspec *);
13830	int        timer_getoverrun(timer_t);
13831	int        timer_settime(timer_t, int, const struct itimerspec *restrict,
13832	struct itimerspec *restrict);
13833 CX	void       tzset(void);
13834	



13835 The following shall be declared as variables:

```
13836 XSI extern int daylight;
13837 extern long timezone;
13838 CX extern char *tzname[];
13839
```

13840 CX Inclusion of the <time.h> header may make visible all symbols from the <signal.h> header.

#### 13841 APPLICATION USAGE

13842 The range [0,60] for *tm\_sec* allows for the occasional leap second.

13843 *tm\_year* is a signed value; therefore, years before 1900 may be represented.

13844 To obtain the number of clock ticks per second returned by the *times()* function, applications  
13845 should call *sysconf(\_SC\_CLK\_TCK)*.

#### 13846 RATIONALE

13847 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of  
13848 UTC does not permit double leap seconds, so all mention of double leap seconds has been  
13849 removed, and the range shortened from the former [0,61] seconds seen in previous versions of  
13850 POSIX.

#### 13851 FUTURE DIRECTIONS

13852 None.

#### 13853 SEE ALSO

13854 <signal.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *asctime()*,  
13855 *clock()*, *clock\_getcpuclockid()*, *clock\_getres()*, *clock\_nanosleep()*, *ctime()*, *difftime()*, *getdate()*,  
13856 *gmtime()*, *localtime()*, *mktime()*, *nanosleep()*, *strptime()*, *sysconf()*, *time()*, *timer\_create()*,  
13857 *timer\_delete()*, *timer\_getoverrun()*, *tzname*, *tzset()*, *utime()*

#### 13858 CHANGE HISTORY

13859 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 13860 Issue 5

13861 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
13862 Threads Extension.

#### 13863 Issue 6

13864 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid\_t**  
13865 and **timer\_t** have been described.

13866 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 13867 • The POSIX timer-related functions are marked as part of the Timers option.

13868 The symbolic name CLK\_TCK is removed. Application usage is added describing how its  
13869 equivalent functionality can be obtained using *sysconf()*.

13870 The *clock\_getcpuclockid()* function and manifest constants CLOCK\_PROCESS\_CPUTIME\_ID and  
13871 CLOCK\_THREAD\_CPUTIME\_ID are added for alignment with IEEE Std 1003.1d-1999.

13872 The manifest constant CLOCK\_MONOTONIC and the *clock\_nanosleep()* function are added for  
13873 alignment with IEEE Std 1003.1j-2000.

13874 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- 13875 • The range for seconds is changed from [0,61] to [0,60].
- 13876 • The **restrict** keyword is added to the prototypes for *asctime\_r()*, *gmtime\_r()*, *localtime\_r()*,  
13877 *strptime()*, *strptime()*, *timer\_create()*, and *timer\_settime()*.

13878 IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the  
13879 **<signal.h>** header may be made visible when the **<time.h>** header is included.  
13880 Extensions beyond the ISO C standard are marked.

13881 **NAME**

13882       trace.h — tracing

13883 **SYNOPSIS**

13884 TRC     #include <trace.h>

13885

13886 **DESCRIPTION**

13887       The <trace.h> header shall define the **posix\_trace\_event\_info** structure that includes at least the  
13888       following members:

13889	trace_event_id_t	posix_event_id
13890	pid_t	posix_pid
13891	void	*posix_prog_address
13892	int	posix_truncation_status
13893	struct timespec	posix_timestamp
13894 THR	pthread_t	posix_thread_id

13895

13896       The <trace.h> header shall define the **posix\_trace\_status\_info** structure that includes at least the  
13897       following members:

13898	int	posix_stream_status
13899	int	posix_stream_full_status
13900	int	posix_stream_overrun_status
13901 TRL	int	posix_stream_flush_status
13902	int	posix_stream_flush_error
13903	int	posix_log_overrun_status
13904	int	posix_log_full_status

13905

13906       The <trace.h> header shall define the following symbols:

13907	POSIX_TRACE_ALL_EVENTS
13908 TRL	POSIX_TRACE_APPEND
13909 TRI	POSIX_TRACE_CLOSE_FOR_CHILD
13910 TEF	POSIX_TRACE_FILTER
13911 TRL	POSIX_TRACE_FLUSH
13912	POSIX_TRACE_FLUSH_START
13913	POSIX_TRACE_FLUSH_STOP
13914	POSIX_TRACE_FLUSHING
13915	POSIX_TRACE_FULL
13916	POSIX_TRACE_LOOP
13917	POSIX_TRACE_NO_OVERRUN
13918 TRL	POSIX_TRACE_NOT_FLUSHING
13919	POSIX_TRACE_NOT_FULL
13920 TRI	POSIX_TRACE_INHERITED
13921	POSIX_TRACE_NOT_TRUNCATED
13922	POSIX_TRACE_OVERFLOW
13923	POSIX_TRACE_OVERRUN
13924	POSIX_TRACE_RESUME
13925	POSIX_TRACE_RUNNING
13926	POSIX_TRACE_START
13927	POSIX_TRACE_STOP
13928	POSIX_TRACE_SUSPENDED
13929	POSIX_TRACE_SYSTEM_EVENTS

```

13930     POSIX_TRACE_TRUNCATED_READ
13931     POSIX_TRACE_TRUNCATED_RECORD
13932     POSIX_TRACE_UNNAMED_USER_EVENT
13933     POSIX_TRACE_UNTIL_FULL
13934     POSIX_TRACE_WOPID_EVENTS

```

13935 The following types shall be defined as described in <sys/types.h>:

```

13936     trace_attr_t
13937     trace_id_t
13938     trace_event_id_t
13939 TEF trace_event_set_t
13940

```

13941 The following shall be declared as functions and may also be defined as macros. Function  
13942 prototypes shall be provided.

```

13943     int  posix_trace_attr_destroy(trace_attr_t *);
13944     int  posix_trace_attr_getclockres(const trace_attr_t *,
13945         struct timespec *);
13946     int  posix_trace_attr_getcreatetime(const trace_attr_t *,
13947         struct timespec *);
13948     int  posix_trace_attr_getgenversion(const trace_attr_t *, char *);
13949 TRI int  posix_trace_attr_getinherited(const trace_attr_t *restrict,
13950     int *restrict);
13951 TRL int  posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
13952     int *restrict);
13953     int  posix_trace_attr_getlogsize(const trace_attr_t *restrict,
13954         size_t *restrict);
13955     int  posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
13956         size_t *restrict);
13957     int  posix_trace_attr_getmaxsystemeventsz(const trace_attr_t *restrict,
13958         size_t *restrict);
13959     int  posix_trace_attr_getmaxusereventsiz(const trace_attr_t *restrict,
13960         size_t, size_t *restrict);
13961     int  posix_trace_attr_getname(const trace_attr_t *, char *);
13962     int  posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
13963         int *restrict);
13964     int  posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
13965         size_t *restrict);
13966     int  posix_trace_attr_init(trace_attr_t *);
13967 TRI int  posix_trace_attr_setinherited(trace_attr_t *, int);
13968 TRL int  posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
13969     int  posix_trace_attr_setlogsize(trace_attr_t *, size_t);
13970     int  posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
13971     int  posix_trace_attr_setname(trace_attr_t *, const char *);
13972     int  posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
13973     int  posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
13974     int  posix_trace_clear(trace_id_t);
13975 TRL int  posix_trace_close(trace_id_t);
13976     int  posix_trace_create(pid_t, const trace_attr_t *restrict,
13977         trace_id_t *restrict);
13978 TRL int  posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
13979     int, trace_id_t *restrict);

```

```

13980 void posix_trace_event(trace_event_id_t, const void *restrict, size_t);
13981 int posix_trace_eventid_equal(trace_id_t, trace_event_id_t,
13982     trace_event_id_t);
13983 int posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
13984 int posix_trace_eventid_open(const char *restrict,
13985     trace_event_id_t *restrict);
13986 TEF int posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
13987 int posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
13988 int posix_trace_eventset_empty(trace_event_set_t *);
13989 int posix_trace_eventset_fill(trace_event_set_t *, int);
13990 int posix_trace_eventset_ismember(trace_event_id_t,
13991     const trace_event_set_t *restrict, int *restrict);
13992 int posix_trace_eventtypelist_getnext_id(trace_id_t,
13993     trace_event_id_t *restrict, int *restrict);
13994 int posix_trace_eventtypelist_rewind(trace_id_t);
13995 TRL int posix_trace_flush(trace_id_t);
13996 int posix_trace_get_attr(trace_id_t, trace_attr_t *);
13997 TEF int posix_trace_get_filter(trace_id_t, trace_event_set_t *);
13998 int posix_trace_get_status(trace_id_t,
13999     struct posix_trace_status_info *);
14000 int posix_trace_getnext_event(trace_id_t,
14001     struct posix_trace_event_info *restrict, void *restrict,
14002     size_t, size_t *restrict, int *restrict);
14003 TRL int posix_trace_open(int, trace_id_t *);
14004 int posix_trace_rewind(trace_id_t);
14005 TEF int posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
14006 int posix_trace_shutdown(trace_id_t);
14007 int posix_trace_start(trace_id_t);
14008 int posix_trace_stop(trace_id_t);
14009 TMO int posix_trace_timedgetnext_event(trace_id_t,
14010     struct posix_trace_event_info *restrict, void *restrict,
14011     size_t, size_t *restrict, int *restrict,
14012     const struct timespec *restrict);
14013 TEF int posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
14014     trace_event_id_t *restrict);
14015 int posix_trace_trygetnext_event(trace_id_t,
14016     struct posix_trace_event_info *restrict, void *restrict, size_t,
14017     size_t *restrict, int *restrict);

```

1

## 14018 APPLICATION USAGE

14019 None.

## 14020 RATIONALE

14021 None.

## 14022 FUTURE DIRECTIONS

14023 None.

## 14024 SEE ALSO

14025 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.11, Tracing,  
14026 *posix\_trace\_attr\_destroy()*, *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*,  
14027 *posix\_trace\_attr\_getgenversion()*, *posix\_trace\_attr\_getinherited()*, *posix\_trace\_attr\_getlogfullpolicy()*,  
14028 *posix\_trace\_attr\_getlogsize()*, *posix\_trace\_attr\_getmaxdatasize()*,  
14029 *posix\_trace\_attr\_getmaxsystemeventsizesize()*, *posix\_trace\_attr\_getmaxusereventsizesize()*,

```

14030     posix_trace_attr_getname(), posix_trace_attr_getstreamfullpolicy(), posix_trace_attr_getstreamsize(),
14031     posix_trace_attr_init(), posix_trace_attr_setinherited(), posix_trace_attr_setlogfullpolicy(),
14032     posix_trace_attr_setlogsize(), posix_trace_attr_setmaxdatasize(), posix_trace_attr_setname(),
14033     posix_trace_attr_setstreamsize(), posix_trace_attr_setstreamfullpolicy(), posix_trace_clear(),
14034     posix_trace_close(), posix_trace_create(), posix_trace_create_withlog(), posix_trace_event(),
14035     posix_trace_eventid_equal(), posix_trace_eventid_get_name(), posix_trace_eventid_open(),
14036     posix_trace_eventtypelist_getnext_id(), posix_trace_eventtypelist_rewind(),
14037     posix_trace_eventset_add(), posix_trace_eventset_del(), posix_trace_eventset_empty(),
14038     posix_trace_eventset_fill(), posix_trace_eventset_ismember(), posix_trace_flush(),
14039     posix_trace_get_attr(), posix_trace_get_filter(), posix_trace_get_status(), posix_trace_getnext_event(),
14040     posix_trace_open(), posix_trace_rewind(), posix_trace_set_filter(), posix_trace_shutdown(),
14041     posix_trace_start(), posix_trace_stop(), posix_trace_timedgetnext_event(),
14042     posix_trace_trid_eventid_open(), posix_trace_trygetnext_event()

```

#### 14043 CHANGE HISTORY

```

14044     First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

14045     IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/40 is applied, adding the TRL margin 1
14046     eode to the posix_trace_flush() function, for alignment with the System Interfaces volume of 1
14047     IEEE Std 1003.1-2001. 1

```

14048 **NAME**

14049       ucontext.h — user context

14050 **SYNOPSIS**

14051 XSI       #include <ucontext.h>

14052

14053 **DESCRIPTION**

14054       The <ucontext.h> header shall define the **mcontext\_t** type through **typedef**.

14055       The <ucontext.h> header shall define the **ucontext\_t** type as a structure that shall include at least  
14056       the following members:

14057	ucontext_t	*uc_link	Pointer to the context that is resumed
14058			when this context returns.
14059	sigset_t	uc_sigmask	The set of signals that are blocked when this
14060			context is active.
14061	stack_t	uc_stack	The stack used by this context.
14062	mcontext_t	uc_mcontext	A machine-specific representation of the saved
14063			context.

14064       The types **sigset\_t** and **stack\_t** shall be defined as in <signal.h>.

14065       The following shall be declared as functions and may also be defined as macros. Function  
14066       prototypes shall be provided.

14067	OB	int	getcontext(ucontext_t *);	2
14068		void	makecontext(ucontext_t *, void (*)(void), int, ...);	2
14069		int	setcontext(const ucontext_t *);	2
14070		int	swapcontext(ucontext_t *restrict, const ucontext_t *restrict);	2
14071				

14072 **APPLICATION USAGE**

14073       None.

14074 **RATIONALE**

14075       None.

14076 **FUTURE DIRECTIONS**

14077       None.

14078 **SEE ALSO**

14079       <signal.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *getcontext()*, *makecontext()*,  
14080       *sigaction()*, *sigprocmask()*, *sigaltstack()*

14081 **CHANGE HISTORY**

14082       First released in Issue 4, Version 2.

14083 **Issue 6**

14084	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/28 is applied, updating the <i>getcontext()</i> ,	2
14085	<i>makecontext()</i> , <i>setcontext()</i> , and <i>swapcontext()</i> functions to be obsolescent.	2

14086 **NAME**

14087       ulimit.h — ulimit commands

14088 **SYNOPSIS**

14089 XSI       #include &lt;ulimit.h&gt;

14090

14091 **DESCRIPTION**14092       The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

14093       Symbolic constants:

14094       UL\_GETFSIZE    Get maximum file size.

14095       UL\_SETFSIZE    Set maximum file size.

14096       The following shall be declared as a function and may also be defined as a macro. A function  
14097       prototype shall be provided.

14098       long ulimit(int, ...);

14099 **APPLICATION USAGE**

14100       None.

14101 **RATIONALE**

14102       None.

14103 **FUTURE DIRECTIONS**

14104       None.

14105 **SEE ALSO**14106       The System Interfaces volume of IEEE Std 1003.1-2001, *ulimit()*14107 **CHANGE HISTORY**

14108       First released in Issue 3.



14109 **NAME**

14110         unistd.h — standard symbolic constants and types

14111 **SYNOPSIS**

14112         #include <unistd.h>

14113 **DESCRIPTION**

14114         The <unistd.h> header defines miscellaneous symbolic constants and types, and declares  
14115         miscellaneous functions. The actual values of the constants are unspecified except as shown. The  
14116         contents of this header are shown below.

14117         **Version Test Macros**

14118         The following symbolic constants shall be defined:

14119         \_POSIX\_VERSION

14120             Integer value indicating version of IEEE Std 1003.1 (C-language binding) to which the  
14121             implementation conforms. For implementations conforming to IEEE Std 1003.1-2001, the  
14122             value shall be 200112L.

14123         \_POSIX2\_VERSION

14124             Integer value indicating version of the Shell and Utilities volume of IEEE Std 1003.1 to  
14125             which the implementation conforms. For implementations conforming to  
14126             IEEE Std 1003.1-2001, the value shall be 200112L.

14127         The following symbolic constant shall be defined only if the implementation supports the XSI  
14128         option; see Section 2.1.4 (on page 21).

14129 XSI         \_XOPEN\_VERSION

14130             Integer value indicating version of the X/Open Portability Guide to which the  
14131             implementation conforms. The value shall be 600.

14132         **Constants for Options and Option Groups**

14133         The following symbolic constants, if defined in <unistd.h>, shall have a value of −1, 0, or greater,  
14134         unless otherwise specified below. If these are undefined, the *fpathconf()*, *pathconf()*, or *sysconf()*  
14135         functions can be used to determine whether the option is provided for a particular invocation of  
14136         the application.

14137         If a symbolic constant is defined with the value −1, the option is not supported. Headers, data  
14138         types, and function interfaces required only for the option need not be supplied. An application  
14139         that attempts to use anything associated only with the option is considered to be requiring an  
14140         extension.

14141         If a symbolic constant is defined with a value greater than zero, the option shall always be  
14142         supported when the application is executed. All headers, data types, and functions shall be  
14143         present and shall operate as specified.

14144         If a symbolic constant is defined with the value zero, all headers, data types, and functions shall  
14145         be present. The application can check at runtime to see whether the option is supported by  
14146         calling *fpathconf()*, *pathconf()*, or *sysconf()* with the indicated *name* parameter.

14147         Unless explicitly specified otherwise, the behavior of functions associated with an unsupported  
14148         option is unspecified, and an application that uses such functions without first checking  
14149         *fpathconf()*, *pathconf()*, or *sysconf()* is considered to be requiring an extension.

14150         For conformance requirements, refer to Chapter 2 (on page 17).

14151	ADV	<b>_POSIX_ADVISORY_INFO</b>	
14152		The implementation supports the Advisory Information option. If this symbol is defined in	2
14153		<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14154		<i>sysconf()</i> shall either be -1 or 200112L.	2
14155	AIO	<b>_POSIX_ASYNCIO</b>	
14156		The implementation supports the Asynchronous Input and Output option. If this symbol is	2
14157		defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14158		reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14159	BAR	<b>_POSIX_BARRIERS</b>	
14160		The implementation supports the Barriers option. If this symbol is defined in <unistd.h>, it	2
14161		shall be defined to be -1, 0, or 200112L. The value of this symbol reported by <i>sysconf()</i> shall	2
14162		either be -1 or 200112L.	2
14163		<b>_POSIX_CHOWN_RESTRICTED</b>	
14164		The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and	
14165		to changing the group ID of a file only to the effective group ID of the process or to one of	2
14166		its supplementary group IDs. This symbol shall either be undefined or defined with a value	2
14167		other than -1.	2
14168	CS	<b>_POSIX_CLOCK_SELECTION</b>	
14169		The implementation supports the Clock Selection option. If this symbol is defined in	2
14170		<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14171		<i>sysconf()</i> shall either be -1 or 200112L.	2
14172	CPT	<b>_POSIX_CPUTIME</b>	
14173		The implementation supports the Process CPU-Time Clocks option. If this symbol is	2
14174		defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14175		reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14176	FSC	<b>_POSIX_FSYNC</b>	
14177		The implementation supports the File Synchronization option. If this symbol is defined in	2
14178		<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14179		<i>sysconf()</i> shall either be -1 or 200112L.	2
14180		<b>_POSIX_IPV6</b>	1
14181		The implementation supports the IPv6 option. If this symbol is defined in <unistd.h>, it	2
14182		shall be defined to be -1, 0, or 200112L. The value of this symbol reported by <i>sysconf()</i> shall	2
14183		either be -1 or 200112L.	2
14184		<b>_POSIX_JOB_CONTROL</b>	
14185		The implementation supports job control. This symbol shall always be set to a value greater	
14186		than zero.	
14187	MF	<b>_POSIX_MAPPED_FILES</b>	
14188		The implementation supports the Memory Mapped Files option. If this symbol is defined in	2
14189		<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14190		<i>sysconf()</i> shall either be -1 or 200112L.	2
14191	ML	<b>_POSIX_MEMLOCK</b>	
14192		The implementation supports the Process Memory Locking option. If this symbol is defined	2
14193		in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported	2
14194		by <i>sysconf()</i> shall either be -1 or 200112L.	2
14195	MLR	<b>_POSIX_MEMLOCK_RANGE</b>	
14196		The implementation supports the Range Memory Locking option. If this symbol is defined	2
14197		in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported	2

14198	by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14199 MPR	<b>_POSIX_MEMORY_PROTECTION</b>	
14200	The implementation supports the Memory Protection option. If this symbol is defined in	2
14201	<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by	2
14202	<i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14203 MSG	<b>_POSIX_MESSAGE_PASSING</b>	
14204	The implementation supports the Message Passing option. If this symbol is defined in	2
14205	<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by	2
14206	<i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14207 MON	<b>_POSIX_MONOTONIC_CLOCK</b>	
14208	The implementation supports the Monotonic Clock option. If this symbol is defined in	2
14209	<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by	2
14210	<i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14211	<b>_POSIX_NO_TRUNC</b>	
14212	Pathname components longer than <code>{NAME_MAX}</code> generate an error. This symbol shall	2
14213	either be undefined or defined with a value other than <code>-1</code> .	2
14214 PIO	<b>_POSIX_PRIORITIZED_IO</b>	
14215	The implementation supports the Prioritized Input and Output option. If this symbol is	2
14216	defined in <unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14217	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14218 PS	<b>_POSIX_PRIORITY_SCHEDULING</b>	
14219	The implementation supports the Process Scheduling option. If this symbol is defined in	2
14220	<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by	2
14221	<i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14222 RS	<b>_POSIX_RAW_SOCKETS</b>	
14223	The implementation supports the Raw Sockets option. If this symbol is defined in	2
14224	<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by	2
14225	<i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14226 THR	<b>_POSIX_READER_WRITER_LOCKS</b>	
14227	The implementation supports the Read-Write Locks option. This is always set to a value	
14228	greater than zero if the Threads option is supported. If this symbol is defined in <unistd.h>,	2
14229	it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by <i>sysconf()</i>	2
14230	shall either be <code>-1</code> or <code>200112L</code> .	2
14231 RTS	<b>_POSIX_REALTIME_SIGNALS</b>	
14232	The implementation supports the Realtime Signals Extension option. If this symbol is	2
14233	defined in <unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14234	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14235	<b>_POSIX_REGEX</b>	
14236	The implementation supports the Regular Expression Handling option. This symbol shall	
14237	always be set to a value greater than zero.	
14238	<b>_POSIX_SAVED_IDS</b>	
14239	Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always	
14240	be set to a value greater than zero.	
14241 SEM	<b>_POSIX_SEMAPHORES</b>	
14242	The implementation supports the Semaphores option. If this symbol is defined in	2
14243	<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by	2

14244	<i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14245 SHM	<b>_POSIX_SHARED_MEMORY_OBJECTS</b>	
14246	The implementation supports the Shared Memory Objects option. If this symbol is defined	2
14247	in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported	2
14248	by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14249	<b>_POSIX_SHELL</b>	
14250	The implementation supports the POSIX shell. This symbol shall always be set to a value	
14251	greater than zero.	
14252 SPN	<b>_POSIX_SPAWN</b>	
14253	The implementation supports the Spawn option. If this symbol is defined in <b>&lt;unistd.h&gt;</b> , it	2
14254	shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by <i>sysconf()</i> shall	2
14255	either be <code>-1</code> or <code>200112L</code> .	2
14256 SPI	<b>_POSIX_SPIN_LOCKS</b>	
14257	The implementation supports the Spin Locks option. If this symbol is defined in	2
14258	<b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported by	2
14259	<i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14260 SS	<b>_POSIX_SPARADIC_SERVER</b>	
14261	The implementation supports the Process Sporadic Server option. If this symbol is defined	2
14262	in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol reported	2
14263	by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14264 SIO	<b>_POSIX_SYNCHRONIZED_IO</b>	
14265	The implementation supports the Synchronized Input and Output option. If this symbol is	2
14266	defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14267	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14268 TSA	<b>_POSIX_THREAD_ATTR_STACKADDR</b>	
14269	The implementation supports the Thread Stack Address Attribute option. If this symbol is	2
14270	defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14271	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14272 TSS	<b>_POSIX_THREAD_ATTR_STACKSIZE</b>	
14273	The implementation supports the Thread Stack Size Attribute option. If this symbol is	2
14274	defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14275	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14276 TCT	<b>_POSIX_THREAD_CPU_TIME</b>	
14277	The implementation supports the Thread CPU-Time Clocks option. If this symbol is	2
14278	defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14279	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14280 TPI	<b>_POSIX_THREAD_PRIO_INHERIT</b>	
14281	The implementation supports the Thread Priority Inheritance option. If this symbol is	2
14282	defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14283	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14284 TPP	<b>_POSIX_THREAD_PRIO_PROTECT</b>	
14285	The implementation supports the Thread Priority Protection option. If this symbol is	2
14286	defined in <b>&lt;unistd.h&gt;</b> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200112L</code> . The value of this symbol	2
14287	reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200112L</code> .	2
14288 TPS	<b>_POSIX_THREAD_PRIORITY_SCHEDULING</b>	
14289	The implementation supports the Thread Execution Scheduling option. If this symbol is	2

14290	defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14291	reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14292 TSH	<b>_POSIX_THREAD_PROCESS_SHARED</b>	
14293	The implementation supports the Thread Process-Shared Synchronization option. If this	2
14294	symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this	2
14295	symbol reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14296 TSF	<b>_POSIX_THREAD_SAFE_FUNCTIONS</b>	
14297	The implementation supports the Thread-Safe Functions option. If this symbol is defined in	2
14298	<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14299	<i>sysconf()</i> shall either be -1 or 200112L.	2
14300 TSP	<b>_POSIX_THREAD_SPORADIC_SERVER</b>	
14301	The implementation supports the Thread Sporadic Server option. If this symbol is defined	2
14302	in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported	2
14303	by <i>sysconf()</i> shall either be -1 or 200112L.	2
14304 THR	<b>_POSIX_THREADS</b>	
14305	The implementation supports the Threads option. If this symbol is defined in <unistd.h>, it	2
14306	shall be defined to be -1, 0, or 200112L. The value of this symbol reported by <i>sysconf()</i> shall	2
14307	either be -1 or 200112L.	2
14308 TMO	<b>_POSIX_TIMEOUTS</b>	
14309	The implementation supports the Timeouts option. If this symbol is defined in <unistd.h>,	2
14310	it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by <i>sysconf()</i>	2
14311	shall either be -1 or 200112L.	2
14312 TMR	<b>_POSIX_TIMERS</b>	
14313	The implementation supports the Timers option. If this symbol is defined in <unistd.h>, it	2
14314	shall be defined to be -1, 0, or 200112L. The value of this symbol reported by <i>sysconf()</i> shall	2
14315	either be -1 or 200112L.	2
14316 TRC	<b>_POSIX_TRACE</b>	
14317	The implementation supports the Trace option. If this symbol is defined in <unistd.h>, it	2
14318	shall be defined to be -1, 0, or 200112L. The value of this symbol reported by <i>sysconf()</i> shall	2
14319	either be -1 or 200112L.	2
14320 TEF	<b>_POSIX_TRACE_EVENT_FILTER</b>	
14321	The implementation supports the Trace Event Filter option. If this symbol is defined in	2
14322	<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14323	<i>sysconf()</i> shall either be -1 or 200112L.	2
14324 TRI	<b>_POSIX_TRACE_INHERIT</b>	
14325	The implementation supports the Trace Inherit option. If this symbol is defined in	2
14326	<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14327	<i>sysconf()</i> shall either be -1 or 200112L.	2
14328 TRL	<b>_POSIX_TRACE_LOG</b>	
14329	The implementation supports the Trace Log option. If this symbol is defined in <unistd.h>,	2
14330	it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by <i>sysconf()</i>	2
14331	shall either be -1 or 200112L.	2
14332 TYM	<b>_POSIX_TYPED_MEMORY_OBJECTS</b>	
14333	The implementation supports the Typed Memory Objects option. If this symbol is defined	2
14334	in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported	2
14335	by <i>sysconf()</i> shall either be -1 or 200112L.	2

14336	<code>_POSIX_VDISABLE</code>	
14337	This symbol shall be defined to be the value of a character that shall disable terminal special	
14338	character handling as described in <termios.h>. This symbol shall always be set to a value	
14339	other than -1.	
14340	<code>_POSIX2_C_BIND</code>	
14341	The implementation supports the C-Language Binding option. This symbol shall always	
14342	have the value 200112L.	
14343 CD	<code>_POSIX2_C_DEV</code>	
14344	The implementation supports the C-Language Development Utilities option. If this symbol	2
14345	is defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14346	reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14347	<code>_POSIX2_CHAR_TERM</code>	
14348	The implementation supports at least one terminal type.	
14349 FD	<code>_POSIX2_FORT_DEV</code>	
14350	The implementation supports the FORTRAN Development Utilities option. If this symbol	2
14351	is defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14352	reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14353 FR	<code>_POSIX2_FORT_RUN</code>	
14354	The implementation supports the FORTRAN Runtime Utilities option. If this symbol is	2
14355	defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14356	reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14357	<code>_POSIX2_LOCALEDEF</code>	
14358	The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol is	2
14359	defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14360	reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14361 BE	<code>_POSIX2_PBS</code>	
14362	The implementation supports the Batch Environment Services and Utilities option. If this	2
14363	symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this	2
14364	symbol reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14365 BE	<code>_POSIX2_PBS_ACCOUNTING</code>	
14366	The implementation supports the Batch Accounting option. If this symbol is defined in	2
14367	<unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported by	2
14368	<i>sysconf()</i> shall either be -1 or 200112L.	2
14369 BE	<code>_POSIX2_PBS_CHECKPOINT</code>	
14370	The implementation supports the Batch Checkpoint/Restart option. If this symbol is	2
14371	defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14372	reported by <i>sysconf()</i> shall either be -1 or 200112L.	2
14373 BE	<code>_POSIX2_PBS_LOCATE</code>	
14374	The implementation supports the Locate Batch Job Request option. If this symbol is defined	2
14375	in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported	2
14376	by <i>sysconf()</i> shall either be -1 or 200112L.	2
14377 BE	<code>_POSIX2_PBS_MESSAGE</code>	
14378	The implementation supports the Batch Job Message Request option. If this symbol is	2
14379	defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14380	reported by <i>sysconf()</i> shall either be -1 or 200112L.	2

14381	BE	<b>_POSIX2_PBS_TRACK</b>	
14382		The implementation supports the Track Batch Job Request option. If this symbol is defined	2
14383		in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported	2
14384		by <i>sysconf</i> () shall either be -1 or 200112L.	2
14385	SD	<b>_POSIX2_SW_DEV</b>	
14386		The implementation supports the Software Development Utilities option. If this symbol is	2
14387		defined in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol	2
14388		reported by <i>sysconf</i> () shall either be -1 or 200112L.	2
14389	UP	<b>_POSIX2_UPE</b>	
14390		The implementation supports the User Portability Utilities option. If this symbol is defined	2
14391		in <unistd.h>, it shall be defined to be -1, 0, or 200112L. The value of this symbol reported	2
14392		by <i>sysconf</i> () shall either be -1 or 200112L.	2
14393		<b>_POSIX_V6_ILP32_OFF32</b>	2
14394		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> ,	
14395		<b>pointer</b> , and <b>off_t</b> types.	
14396		<b>_POSIX_V6_ILP32_OFFBIG</b>	2
14397		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> ,	
14398		and <b>pointer</b> types and an <b>off_t</b> type using at least 64 bits.	
14399		<b>_POSIX_V6_LP64_OFF64</b>	2
14400		The implementation provides a C-language compilation environment with 32-bit <b>int</b> and	
14401		64-bit <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.	
14402		<b>_POSIX_V6_LPBIG_OFFBIG</b>	2
14403		The implementation provides a C-language compilation environment with an <b>int</b> type	
14404		using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.	
14405	XSI	<b>_XBS5_ILP32_OFF32 (LEGACY)</b>	
14406		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> ,	
14407		<b>pointer</b> , and <b>off_t</b> types.	
14408	XSI	<b>_XBS5_ILP32_OFFBIG (LEGACY)</b>	
14409		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> ,	
14410		and <b>pointer</b> types and an <b>off_t</b> type using at least 64 bits.	
14411	XSI	<b>_XBS5_LP64_OFF64 (LEGACY)</b>	
14412		The implementation provides a C-language compilation environment with 32-bit <b>int</b> and	
14413		64-bit <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.	
14414	XSI	<b>_XBS5_LPBIG_OFFBIG (LEGACY)</b>	
14415		The implementation provides a C-language compilation environment with an <b>int</b> type	
14416		using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.	
14417	XSI	<b>_XOPEN_CRYPT</b>	
14418		The implementation supports the X/Open Encryption Option Group.	
14419		<b>_XOPEN_ENH_I18N</b>	
14420		The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option	
14421		Group. This symbol shall always be set to a value other than -1.	
14422		<b>_XOPEN_LEGACY</b>	
14423		The implementation supports the Legacy Option Group.	
14424		<b>_XOPEN_REALTIME</b>	
14425		The implementation supports the X/Open Realtime Option Group.	

14426	<b>_XOPEN_REALTIME_THREADS</b>
14427	The implementation supports the X/Open Realtime Threads Option Group.
14428	<b>_XOPEN_SHM</b>
14429	The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This
14430	symbol shall always be set to a value other than -1.
14431	<b>_XOPEN_STREAMS</b>
14432	The implementation supports the XSI STREAMS Option Group.
14433 XSI	<b>_XOPEN_UNIX</b>
14434	The implementation supports the XSI extension.
14435	<b>Execution-Time Symbolic Constants</b>
14436	If any of the following constants are not defined in the <unistd.h> header, the value shall vary
14437	depending on the file to which it is applied.
14438	If any of the following constants are defined to have value -1 in the <unistd.h> header, the
14439	implementation shall not provide the option on any file; if any are defined to have a value other
14440	than -1 in the <unistd.h> header, the implementation shall provide the option on all applicable
14441	files.
14442	All of the following constants, whether defined in <unistd.h> or not, may be queried with
14443	respect to a specific file using the <i>pathconf()</i> or <i>fpathconf()</i> functions:
14444	<b>_POSIX_ASYNC_IO</b>
14445	Asynchronous input or output operations may be performed for the associated file.
14446	<b>_POSIX_PRIO_IO</b>
14447	Prioritized input or output operations may be performed for the associated file.
14448	<b>_POSIX_SYNC_IO</b>
14449	Synchronized input or output operations may be performed for the associated file.
14450	<b>Constants for Functions</b>
14451	The following symbolic constant shall be defined:
14452	<b>NULL</b> Null pointer
14453	The following symbolic constants shall be defined for the <i>access()</i> function:
14454	<b>F_OK</b> Test for existence of file.
14455	<b>R_OK</b> Test for read permission.
14456	<b>W_OK</b> Test for write permission.
14457	<b>X_OK</b> Test for execute (search) permission.
14458	The constants F_OK, R_OK, W_OK, and X_OK and the expressions <i>R_OK   W_OK</i> , <i>R_OK   X_OK</i> ,
14459	and <i>R_OK   W_OK   X_OK</i> shall all have distinct values.
14460	The following symbolic constants shall be defined for the <i>confstr()</i> function:
14461	<b>_CS_PATH</b>
14462	This is the value for the <i>PATH</i> environment variable that finds all standard utilities.
14463	<b>_CS_POSIX_V6_ILP32_OFF32_CFLAGS</b>
14464	If <i>sysconf(_SC_V6_ILP32_OFF32)</i> returns -1, the meaning of this value is unspecified.
14465	Otherwise, this value is the set of initial options to be given to the <i>c99</i> utility to build an



14466 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.

14467 **\_CS\_POSIX\_V6\_ILP32\_OFF32\_LDFLAGS**  
 14468 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.  
 14469 Otherwise, this value is the set of final options to be given to the `c99` utility to build an  
 14470 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.

14471 **\_CS\_POSIX\_V6\_ILP32\_OFF32\_LIBS**  
 14472 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.  
 14473 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an  
 14474 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.

14475 **\_CS\_POSIX\_V6\_ILP32\_OFFBIG\_CFLAGS**  
 14476 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14477 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an  
 14478 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an  
 14479 **off\_t** type using at least 64 bits.

14480 **\_CS\_POSIX\_V6\_ILP32\_OFFBIG\_LDFLAGS**  
 14481 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14482 Otherwise, this value is the set of final options to be given to the `c99` utility to build an  
 14483 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an  
 14484 **off\_t** type using at least 64 bits.

14485 **\_CS\_POSIX\_V6\_ILP32\_OFFBIG\_LIBS**  
 14486 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14487 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an  
 14488 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an  
 14489 **off\_t** type using at least 64 bits.

14490 **\_CS\_POSIX\_V6\_LP64\_OFF64\_CFLAGS**  
 14491 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.  
 14492 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an 1  
 14493 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off\_t** 1  
 14494 types.

14495 **\_CS\_POSIX\_V6\_LP64\_OFF64\_LDFLAGS**  
 14496 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.  
 14497 Otherwise, this value is the set of final options to be given to the `c99` utility to build an 1  
 14498 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off\_t** 1  
 14499 types.

14500 **\_CS\_POSIX\_V6\_LP64\_OFF64\_LIBS**  
 14501 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.  
 14502 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an 1  
 14503 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off\_t** 1  
 14504 types.

14505 **\_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_CFLAGS**  
 14506 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14507 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an  
 14508 application using a programming model with an **int** type using at least 32 bits and **long**,  
 14509 **pointer**, and **off\_t** types using at least 64 bits.

14510 **\_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LDFLAGS**  
 14511 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 14512 Otherwise, this value is the set of final options to be given to the `c99` utility to build an

14513 application using a programming model with an **int** type using at least 32 bits and **long**,  
 14514 **pointer**, and **off\_t** types using at least 64 bits.

14515 **\_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LIBS**

14516 If *sysconf*(*\_SC\_V6\_LPBIG\_OFFBIG*) returns **-1**, the meaning of this value is unspecified.  
 14517 Otherwise, this value is the set of libraries to be given to the *c99* utility to build an  
 14518 application using a programming model with an **int** type using at least 32 bits and **long**,  
 14519 **pointer**, and **off\_t** types using at least 64 bits.

14520 **\_CS\_POSIX\_V6\_WIDTH\_RESTRICTED\_ENVS**

14521 This value is a <newline>-separated list of names of programming environments supported  
 14522 by the implementation in which the widths of the **blksize\_t**, **cc\_t**, **mode\_t**, **nfds\_t**, **pid\_t**,  
 14523 **ptrdiff\_t**, **size\_t**, **speed\_t**, **ssize\_t**, **suseconds\_t**, **tcflag\_t**, **useconds\_t**, **wchar\_t**, and **wint\_t**  
 14524 types are no greater than the width of type **long**. The format of each name shall be suitable  
 14525 for use with the *getconf* **-v** option.

2  
2

14526 XSI The following symbolic constants are reserved for compatibility with Issue 5:

14527 **\_CS\_XBS5\_ILP32\_OFF32\_CFLAGS (LEGACY)**  
 14528 **\_CS\_XBS5\_ILP32\_OFF32\_LDFLAGS (LEGACY)**  
 14529 **\_CS\_XBS5\_ILP32\_OFF32\_LIBS (LEGACY)**  
 14530 **\_CS\_XBS5\_ILP32\_OFF32\_LINTFLAGS (LEGACY)**  
 14531 **\_CS\_XBS5\_ILP32\_OFFBIG\_CFLAGS (LEGACY)**  
 14532 **\_CS\_XBS5\_ILP32\_OFFBIG\_LDFLAGS (LEGACY)**  
 14533 **\_CS\_XBS5\_ILP32\_OFFBIG\_LIBS (LEGACY)**  
 14534 **\_CS\_XBS5\_ILP32\_OFFBIG\_LINTFLAGS (LEGACY)**  
 14535 **\_CS\_XBS5\_LP64\_OFF64\_CFLAGS (LEGACY)**  
 14536 **\_CS\_XBS5\_LP64\_OFF64\_LDFLAGS (LEGACY)**  
 14537 **\_CS\_XBS5\_LP64\_OFF64\_LIBS (LEGACY)**  
 14538 **\_CS\_XBS5\_LP64\_OFF64\_LINTFLAGS (LEGACY)**  
 14539 **\_CS\_XBS5\_LPBIG\_OFFBIG\_CFLAGS (LEGACY)**  
 14540 **\_CS\_XBS5\_LPBIG\_OFFBIG\_LDFLAGS (LEGACY)**  
 14541 **\_CS\_XBS5\_LPBIG\_OFFBIG\_LIBS (LEGACY)**  
 14542 **\_CS\_XBS5\_LPBIG\_OFFBIG\_LINTFLAGS (LEGACY)**  
 14543

14544 The following symbolic constants shall be defined for the *lseek*() and *fcntl*() functions and shall  
 14545 have distinct values:

14546 **SEEK\_CUR** Set file offset to current plus *offset*.  
 14547 **SEEK\_END** Set file offset to EOF plus *offset*.  
 14548 **SEEK\_SET** Set file offset to *offset*.

14549 The following symbolic constants shall be defined as possible values for the *function* argument  
 14550 to the *lockf*() function:

14551 **F\_LOCK** Lock a section for exclusive use.  
 14552 **F\_TEST** Test section for locks by other processes.  
 14553 **F\_TLOCK** Test and lock a section for exclusive use.  
 14554 **F\_ULOCK** Unlock locked sections.

14555	The following symbolic constants shall be defined for <i>pathconf()</i> :	
14556	_PC_2_SYMLINKS	2
14557	_PC_ALLOC_SIZE_MIN	1
14558	_PC_ASYNC_IO	1
14559	_PC_CHOWN_RESTRICTED	1
14560	_PC_FILESIZEBITS	
14561	_PC_LINK_MAX	
14562	_PC_MAX_CANON	
14563	_PC_MAX_INPUT	
14564	_PC_NAME_MAX	
14565	_PC_NO_TRUNC	
14566	_PC_PATH_MAX	
14567	_PC_PIPE_BUF	
14568	_PC_PRIO_IO	
14569	_PC_REC_INCR_XFER_SIZE	1
14570	_PC_REC_MIN_XFER_SIZE	1
14571	_PC_REC_XFER_ALIGN	
14572	_PC_SYMLINK_MAX	1
14573	_PC_SYNC_IO	1
14574	_PC_VDISABLE	
14575	The following symbolic constants shall be defined for <i>sysconf()</i> :	
14576	_SC_2_C_BIND	
14577	_SC_2_C_DEV	
14578	_SC_2_CHAR_TERM	2
14579	_SC_2_FORT_DEV	
14580	_SC_2_FORT_RUN	
14581	_SC_2_LOCALEDEF	
14582	_SC_2_PBS	
14583	_SC_2_PBS_ACCOUNTING	
14584	_SC_2_PBS_CHECKPOINT	
14585	_SC_2_PBS_LOCATE	
14586	_SC_2_PBS_MESSAGE	
14587	_SC_2_PBS_TRACK	
14588	_SC_2_SW_DEV	
14589	_SC_2_UPE	
14590	_SC_2_VERSION	
14591	_SC_ADVISORY_INFO	
14592	_SC_AIO_LISTIO_MAX	
14593	_SC_AIO_MAX	
14594	_SC_AIO_PRIO_DELTA_MAX	
14595	_SC_ARG_MAX	
14596	_SC_ASYNCHRONOUS_IO	
14597	_SC_ATEXIT_MAX	1
14598	_SC_BARRIERS	1
14599	_SC_BC_BASE_MAX	1
14600	_SC_BC_DIM_MAX	
14601	_SC_BC_SCALE_MAX	
14602	_SC_BC_STRING_MAX	
14603	_SC_CHILD_MAX	
14604	_SC_CLK_TCK	
14605	_SC_CLOCK_SELECTION	1

14606	_SC_COLL_WEIGHTS_MAX	1
14607	_SC_CPUTIME	
14608	_SC_DELAYTIMER_MAX	
14609	_SC_EXPR_NEST_MAX	
14610	_SC_FSYNC	2
14611	_SC_GETGR_R_SIZE_MAX	
14612	_SC_GETPW_R_SIZE_MAX	
14613	_SC_HOST_NAME_MAX	
14614	_SC_IOV_MAX	1
14615	_SC_IPV6	1
14616	_SC_JOB_CONTROL	1
14617	_SC_LINE_MAX	
14618	_SC_LOGIN_NAME_MAX	
14619	_SC_MAPPED_FILES	
14620	_SC_MEMLOCK	
14621	_SC_MEMLOCK_RANGE	
14622	_SC_MEMORY_PROTECTION	
14623	_SC_MESSAGE_PASSING	
14624	_SC_MONOTONIC_CLOCK	1
14625	_SC_MQ_OPEN_MAX	1
14626	_SC_MQ_PRIO_MAX	
14627	_SC_NGROUPS_MAX	
14628	_SC_OPEN_MAX	
14629	_SC_PAGE_SIZE	1
14630	_SC_PAGESIZE	
14631	_SC_PRIORITIZED_IO	1
14632	_SC_PRIORITY_SCHEDULING	
14633	_SC_RAW_SOCKETS	1
14634	_SC_RE_DUP_MAX	1
14635	_SC_READER_WRITER_LOCKS	1
14636	_SC_REALTIME_SIGNALS	1
14637	_SC_REGEX	
14638	_SC_RTSIG_MAX	
14639	_SC_SAVED_IDS	
14640	_SC_SEM_NSEMS_MAX	1
14641	_SC_SEM_VALUE_MAX	
14642	_SC_SEMAPHORES	1
14643	_SC_SHARED_MEMORY_OBJECTS	1
14644	_SC_SHELL	
14645	_SC_SIGQUEUE_MAX	
14646	_SC_SPAWN	
14647	_SC_SPIN_LOCKS	1
14648	_SC_SPORADIC_SERVER	1
14649	_SC_SS_REPL_MAX	2
14650	_SC_STREAM_MAX	
14651	_SC_SYMLOOP_MAX	1
14652	_SC_SYNCHRONIZED_IO	1
14653	_SC_THREAD_ATTR_STACKADDR	
14654	_SC_THREAD_ATTR_STACKSIZE	
14655	_SC_THREAD_CPUTIME	
14656	_SC_THREAD_DESTRUCTOR_ITERATIONS	
14657	_SC_THREAD_KEYS_MAX	

14658	_SC_THREAD_PRIO_INHERIT	
14659	_SC_THREAD_PRIO_PROTECT	
14660	_SC_THREAD_PRIORITY_SCHEDULING	
14661	_SC_THREAD_PROCESS_SHARED	
14662	_SC_THREAD_SAFE_FUNCTIONS	
14663	_SC_THREAD_SPORADIC_SERVER	
14664	_SC_THREAD_STACK_MIN	
14665	_SC_THREAD_THREADS_MAX	
14666	_SC_THREADS	1
14667	_SC_TIMEOUTS	1
14668	_SC_TIMER_MAX	1
14669	_SC_TIMERS	1
14670	_SC_TRACE	1
14671	_SC_TRACE_EVENT_FILTER	1
14672	_SC_TRACE_EVENT_NAME_MAX	2
14673	_SC_TRACE_INHERIT	1
14674	_SC_TRACE_LOG	1
14675	_SC_TRACE_NAME_MAX	2
14676	_SC_TRACE_SYS_MAX	2
14677	_SC_TRACE_USER_EVENT_MAX	2
14678	_SC_TTY_NAME_MAX	1
14679	_SC_TYPED_MEMORY_OBJECTS	1
14680	_SC_TZNAME_MAX	1
14681	_SC_V6_ILP32_OFF32	
14682	_SC_V6_ILP32_OFFBIG	
14683	_SC_V6_LP64_OFF64	
14684	_SC_V6_LPBIG_OFFBIG	
14685	_SC_VERSION	
14686	_SC_XBS5_ILP32_OFF32 (LEGACY)	1
14687	_SC_XBS5_ILP32_OFFBIG (LEGACY)	
14688	_SC_XBS5_LP64_OFF64 (LEGACY)	
14689	_SC_XBS5_LPBIG_OFFBIG (LEGACY)	
14690	_SC_XOPEN_CRYPT	
14691	_SC_XOPEN_ENH_I18N	
14692	_SC_XOPEN_LEGACY	
14693	_SC_XOPEN_REALTIME	
14694	_SC_XOPEN_REALTIME_THREADS	
14695	_SC_XOPEN_SHM	
14696	_SC_XOPEN_STREAMS	
14697	_SC_XOPEN_UNIX	
14698	_SC_XOPEN_VERSION	
14699	The two constants _SC_PAGESIZE and _SC_PAGE_SIZE may be defined to have the same	
14700	value.	
14701	The following symbolic constants shall be defined for file streams:	
14702	STDERR_FILENO	File number of <i>stderr</i> ; 2.
14703	STDIN_FILENO	File number of <i>stdin</i> ; 0.
14704	STDOUT_FILENO	File number of <i>stdout</i> ; 1.

14705      **Type Definitions**

14706      The **size\_t**, **ssize\_t**, **uid\_t**, **gid\_t**, **off\_t**, **pid\_t**, and **useconds\_t** types shall be defined as described  
 14707      in <sys/types.h>.

14708      The **intptr\_t** type shall be defined as described in <inttypes.h>.

14709      **Declarations**

14710      The following shall be declared as functions and may also be defined as macros. Function  
 14711      prototypes shall be provided.

```

14712      int            access(const char *, int);
14713      unsigned      alarm(unsigned);
14714      int            chdir(const char *);
14715      int            chown(const char *, uid_t, gid_t);
14716      int            close(int);
14717      size_t        confstr(int, char *, size_t);
14718 XSI      char        *crypt(const char *, const char *);
14719      char        *ctermid(char *);
14720      int            dup(int);
14721      int            dup2(int, int);
14722 XSI      void        encrypt(char[64], int);
14723      int            execl(const char *, const char *, ...);
14724      int            execle(const char *, const char *, ...);
14725      int            execlp(const char *, const char *, ...);
14726      int            execv(const char *, char *const []);
14727      int            execve(const char *, char *const [], char *const []);
14728      int            execvp(const char *, char *const []);
14729      void           _exit(int);
14730      int            fchown(int, uid_t, gid_t);
14731 XSI      int           fchdir(int);
14732 SIO      int           fdatsync(int);
14733      pid_t        fork(void);
14734      long         fpathconf(int, int);
14735 FSC      int           fsync(int);
14736      int           ftruncate(int, off_t);
14737      char        *getcwd(char *, size_t);
14738      gid_t        getegid(void);
14739      uid_t        geteuid(void);
14740      gid_t        getgid(void);
14741      int           getgroups(int, gid_t []);
14742 XSI      long        gethostid(void);
14743      int           gethostname(char *, size_t);
14744      char        *getlogin(void);
14745      int           getlogin_r(char *, size_t);
14746      int           getopt(int, char * const [], const char *);
14747 XSI      pid_t        getpgid(pid_t);
14748      pid_t        getpgrp(void);
14749      pid_t        getpid(void);
14750      pid_t        getppid(void);
14751 XSI      pid_t        getsid(pid_t);
14752      uid_t        getuid(void);
14753 XSI

```

1

14754	char	*getwd(char *); ( <b>LEGACY</b> )	
14755	int	isatty(int);	
14756 XSI	int	lchown(const char *, uid_t, gid_t);	
14757	int	link(const char *, const char *);	
14758 XSI	int	lockf(int, int, off_t);	
14759	off_t	lseek(int, off_t, int);	
14760 XSI	int	nice(int);	
14761	long	pathconf(const char *, int);	
14762	int	pause(void);	
14763	int	pipe(int [2]);	
14764 XSI	ssize_t	pread(int, void *, size_t, off_t);	
14765	ssize_t	pwrite(int, const void *, size_t, off_t);	
14766	ssize_t	read(int, void *, size_t);	
14767	ssize_t	readlink(const char *restrict, char *restrict, size_t);	
14768	int	rmdir(const char *);	
14769	int	setegid(gid_t);	
14770	int	seteuid(uid_t);	
14771	int	setgid(gid_t);	
14772	int	setpgid(pid_t, pid_t);	
14773 XSI	pid_t	setpgrp(void);	
14774	int	setregid(gid_t, gid_t);	
14775	int	setreuid(uid_t, uid_t);	
14776	pid_t	setsid(void);	
14777	int	setuid(uid_t);	
14778	unsigned	sleep(unsigned);	
14779 XSI	void	swab(const void *restrict, void *restrict, ssize_t);	
14780	int	symlink(const char *, const char *);	2
14781 XSI	void	sync(void);	2
14782	long	sysconf(int);	
14783	pid_t	tcgetpgrp(int);	
14784	int	tcsetpgrp(int, pid_t);	
14785 XSI	int	truncate(const char *, off_t);	
14786	char	*ttyname(int);	
14787	int	ttyname_r(int, char *, size_t);	
14788 XSI	useconds_t	ualarm(useconds_t, useconds_t);	
14789	int	unlink(const char *);	
14790 XSI	int	usleep(useconds_t);	
14791	pid_t	vfork(void);	
14792	ssize_t	write(int, const void *, size_t);	

14793 Implementations may also include the *pthread\_atfork()* prototype as defined in <pthread.h> (on  
14794 page 290).

14795 The following external variables shall be declared:

14796 extern char \*optarg;  
14797 extern int optind, opterr, optopt;

14798 **APPLICATION USAGE**

14799 IEEE Std 1003.1-2001 only describes the behavior of systems that claim conformance to it.  
 14800 However, application developers who want to write applications that adapt to other versions of  
 14801 IEEE Std 1003.1 (or to systems that do not conform to any POSIX standard) may find it useful to  
 14802 code them so as to conditionally compile different code depending on the value of  
 14803 `_POSIX_VERSION`, for example:

```
14804 #if _POSIX_VERSION >= 200112L
14805 /* Use the newer function that copes with large files. */
14806 off_t pos=ftello(fp);
14807 #else
14808 /* Either this is an old version of POSIX, or _POSIX_VERSION is
14809    not even defined, so use the traditional function. */
14810 long pos=ftell(fp);
14811 #endif
```

14812 Earlier versions of IEEE Std 1003.1 and of the Single UNIX Specification can be identified by the  
 14813 following macros:

14814 POSIX.1-1988 standard  
 14815 `_POSIX_VERSION==198808L`

14816 POSIX.1-1990 standard  
 14817 `_POSIX_VERSION==199009L`

14818 ISO POSIX-1:1996 standard  
 14819 `_POSIX_VERSION==199506L`

14820 Single UNIX Specification, Version 1  
 14821 `_XOPEN_UNIX` and `_XOPEN_VERSION==4`

14822 Single UNIX Specification, Version 2  
 14823 `_XOPEN_UNIX` and `_XOPEN_VERSION==500`

14824 IEEE Std 1003.1-2001 does not make any attempt to define application binary interaction with  
 14825 the underlying operating system. However, application developers may find it useful to query  
 14826 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the  
 14827 operating system supports the necessary functionality as in the following program fragment:

```
14828 if (sysconf(_SC_VERSION) < 200112L) {
14829     fprintf(stderr, "POSIX.1-2001 system required, terminating \n");
14830     exit(1);
14831 }
```

14832 New applications should not use `_XOPEN_SHM` or `_XOPEN_ENH_I18N`.

1

14833 **RATIONALE**

14834 As IEEE Std 1003.1-2001 evolved, certain options became sufficiently standardized that it was  
 14835 concluded that simply requiring one of the option choices was simpler than retaining the option.  
 14836 However, for backwards-compatibility, the option flags (with required constant values) are  
 14837 retained.



14838 **Version Test Macros**

14839 The standard developers considered altering the definition of `_POSIX_VERSION` and removing  
14840 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed  
14841 by some to be minimal, and since the implementation of the functionality is potentially  
14842 problematic. However, they recognized that support for existing application binaries is a  
14843 concern to manufacturers, application developers, and the users of implementations conforming  
14844 to IEEE Std 1003.1-2001.

14845 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide  
14846 the greatest degree of imaginable utility to the application developer or user, it is arguably better  
14847 than a **core** file or some other equally obscure result. (It is also possible for implementations to  
14848 encode and recognize application binaries compiled in various POSIX.1-conforming  
14849 environments, and modify the semantics of the underlying system to conform to the  
14850 expectations of the application.) For the reasons outlined in the preceding paragraphs and in the  
14851 APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`  
14852 and `_SC_VERSION` functionality.

14853 **Compile-Time Symbolic Constants for System-Wide Options**

14854 IEEE Std 1003.1-2001 now includes support in certain areas for the newly adopted policy  
14855 governing options and stubs.

14856 This policy provides flexibility for implementations in how they support options. It also  
14857 specifies how conforming applications can adapt to different implementations that support  
14858 different sets of options. It allows the following:

- 14859 1. If an implementation has no interest in supporting an option, it does not have to provide  
14860 anything associated with that option beyond the announcement that it does not support it.
- 14861 2. An implementation can support a partial or incompatible version of an option (as a non-  
14862 standard extension) as long as it does not claim to support the option.
- 14863 3. An application can determine whether the option is supported. A strictly conforming  
14864 application must check this announcement mechanism before first using anything  
14865 associated with the option.

14866 There is an important implication of this policy. IEEE Std 1003.1-2001 cannot dictate the  
14867 behavior of interfaces associated with an option when the implementation does not claim to  
14868 support the option. In particular, it cannot require that a function associated with an  
14869 unsupported option will fail if it does not perform as specified. However, this policy does not  
14870 prevent a standard from requiring certain functions to always be present, but that they shall  
14871 always fail on some implementations. The `setpgid()` function in the POSIX.1-1990 standard, for  
14872 example, is considered appropriate.

14873 The POSIX standards include various options, and the C-language binding support for an option  
14874 implies that the implementation must supply data types and function interfaces. An application  
14875 must be able to discover whether the implementation supports each option.

14876 Any application must consider the following three cases for each option:

- 14877 1. Option never supported.

14878 The implementation advertises at compile time that the option will never be supported. In  
14879 this case, it is not necessary for the implementation to supply any of the data types or  
14880 function interfaces that are provided only as part of the option. The implementation might  
14881 provide data types and functions that are similar to those defined by IEEE Std 1003.1-2001,  
14882 but there is no guarantee for any particular behavior.

- 14883           2. Option always supported.
- 14884           The implementation advertises at compile time that the option will always be supported.
- 14885           In this case, all data types and function interfaces shall be available and shall operate as
- 14886           specified.
- 14887           3. Option might or might not be supported.
- 14888           Some implementations might not provide a mechanism to specify support of options at
- 14889           compile time. In addition, the implementation might be unable or unwilling to specify
- 14890           support or non-support at compile time. In either case, any application that might use the
- 14891           option at runtime must be able to compile and execute. The implementation must provide,
- 14892           at compile time, all data types and function interfaces that are necessary to allow this. In
- 14893           this situation, there must be a mechanism that allows the application to query, at runtime,
- 14894           whether the option is supported. If the application attempts to use the option when it is
- 14895           not supported, the result is unspecified unless explicitly specified otherwise in
- 14896           IEEE Std 1003.1-2001.
- 14897 **FUTURE DIRECTIONS**
- 14898           None.
- 14899 **SEE ALSO**
- 14900           <inttypes.h>, <limits.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, <wctype.h>, the System
- 14901           Interfaces volume of IEEE Std 1003.1-2001, *access()*, *alarm()*, *chdir()*, *chown()*, *close()*, *crypt()*,
- 14902           *ctermid()*, *dup()*, *encrypt()*, *environ*, *exec*, *exit()*, *fchdir()*, *fchown()*, *fcntl()*, *fork()*, *fpathconf()*,
- 14903           *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*,
- 14904           *getlogin()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*,
- 14905           *lockf()*, *lseek()*, *nice()*, *pathconf()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setgid()*, *setpgid()*,
- 14906           *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*,
- 14907           *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *ualarm()*, *unlink()*, *usleep()*, *vfork()*, *write()*
- 14908 **CHANGE HISTORY**
- 14909           First released in Issue 1. Derived from Issue 1 of the SVID.
- 14910 **Issue 5**
- 14911           The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
- 14912           Threads Extension.
- 14913           The symbolic constants *\_XOPEN\_REALTIME* and *\_XOPEN\_REALTIME\_THREADS* are added.
- 14914           *\_POSIX2\_C\_BIND*, *\_XOPEN\_ENH\_I18N*, and *\_XOPEN\_SHM* must now be set to a value other
- 14915           than *-1* by a conforming implementation.
- 14916           Large File System extensions are added.
- 14917           The type of the argument to *sbrk()* is changed from *int* to *intptr\_t*.
- 14918           *\_XBS\_* constants are added to the list of constants for Options and Option Groups, to the list of
- 14919           constants for the *confstr()* function, and to the list of constants to the *sysconf()* function. These
- 14920           are all marked EX.
- 14921 **Issue 6**
- 14922           *\_POSIX2\_C\_VERSION* is removed.
- 14923           The Open Group Corrigendum U026/4 is applied, adding the prototype for *fdatasync()*.
- 14924           The Open Group Corrigendum U026/1 is applied, adding the symbols *\_SC\_XOPEN\_LEGACY*,
- 14925           *\_SC\_XOPEN\_REALTIME*, and *\_SC\_XOPEN\_REALTIME\_THREADS*.
- 14926           The symbols *\_XOPEN\_STREAMS* and *\_SC\_XOPEN\_STREAMS* are added to support the XSI
- 14927           STREAMS Option Group.

14928 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in  
 14929 IEEE Std 1003.1-2001.

14930 The legacy symbol `_SC_PASS_MAX` is removed.

14931 The following new requirements on POSIX implementations derive from alignment with the  
 14932 Single UNIX Specification:

- 14933 • The `_CS_POSIX_*` and `_CS_XBS5_*` constants are added for the `confstr()` function.
- 14934 • The `_SC_XBS5_*` constants are added for the `sysconf()` function.
- 14935 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.
- 14936 • The `uid_t`, `gid_t`, `off_t`, `pid_t`, and `useconds_t` types are mandated.

14937 The `gethostname()` prototype is added for sockets.

14938 A new section is added for System-Wide Options.

14939 Function prototypes for `setegid()` and `seteuid()` are added.

14940 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`,  
 14941 `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,  
 14942 `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and `pathconf()` variables are  
 14943 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,  
 14944 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with  
 14945 IEEE Std 1003.1d-1999.

14946 The following are added for alignment with IEEE Std 1003.1j-2000:

- 14947 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,  
 14948 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,  
 14949 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`
- 14950 • `sysconf()` variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,  
 14951 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and  
 14952 `_SC_TYPED_MEMORY_OBJECTS`

14953 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,  
 14954 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are  
 14955 introduced.

14956 The `getwd()` function is marked LEGACY.

14957 The `restrict` keyword is added to the prototypes for `readlink()` and `swab()`.

14958 Constants for options are now harmonized, so when supported they take the year of approval of  
 14959 IEEE Std 1003.1-2001 as the value.

14960 The following are added for alignment with IEEE Std 1003.1q-2000:

- 14961 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,  
 14962 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`
- 14963 • The `sysconf()` symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,  
 14964 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`

14965 The `brk()` and `sbrk()` legacy functions are removed.

14966 The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning  
 14967 information.

14968	The Open Group Base Resolution bwg2001-008 is applied, changing the <i>namelen</i> parameter for	
14969	<i>gethostname()</i> from <b>socklen_t</b> to <b>size_t</b> .	
14970	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing “Thread Stack	1
14971	Address Size” to “Thread Stack Size Attribute”.	1
14972	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the <b>_POSIX_IPV6</b> ,	1
14973	<b>_SC_V6</b> , and <b>_SC_RAW_SOCKETS</b> symbols.	1
14974	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description in	1
14975	“Constants for Functions” for the <b>_CS_POSIX_V6_LP64_OFF64_CFLAGS</b> ,	1
14976	<b>_CS_POSIX_V6_LP64_OFF64_LDFLAGS</b> , and <b>_CS_POSIX_V6_LP64_OFF64_LIBS</b> symbols.	1
14977	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for	1
14978	the <b>_PC*</b> and <b>_SC*</b> constants, since these are mandatory on all implementations.	1
14979	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/23 is applied, adding the	1
14980	<b>_PC_SYMLINK_MAX</b> and <b>_SC_SYMLINK_MAX</b> constants.	1
14981	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/24 is applied, correcting the shading and	1
14982	margin code for the <i>fsync()</i> function.	1
14983	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/25 is applied, adding the following text to	1
14984	the APPLICATION USAGE section: “New applications should not use <b>_XOPEN_SHM</b> or	1
14985	<b>_XOPEN_ENH_I18N</b> .”.	1
14986	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/29 is applied, clarifying the requirements	2
14987	for when constants for Options and Option Groups can be defined or undefined.	2
14988	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/30 is applied, changing the	2
14989	<b>_V6_ILP32_OFF32</b> , <b>_V6_ILP32_OFFBIG</b> , <b>_V6_LP64_OFF64</b> , and <b>_V6_LPBIG_OFFBIG</b> symbols to	2
14990	<b>_POSIX_V6_ILP32_OFF32</b> , <b>_POSIX_V6_ILP32_OFFBIG</b> , <b>_POSIX_V6_LP64_OFF64</b> , and	2
14991	<b>_POSIX_V6_LPBIG_OFFBIG</b> , respectively. This is for consistency with the <i>sysconf()</i> and <i>c99</i>	2
14992	reference pages.	2
14993	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/31 is applied, adding that the format of	2
14994	names of programming environments can be obtained using the <i>getconf -v</i> option.	2
14995	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/32 is applied, deleting the	2
14996	<b>_SC_FILE_LOCKING</b> , <b>_SC_2_C_VERSION</b> , and <b>_SC_XOPEN_XCU_VERSION</b> constants.	2
14997	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/33 is applied, adding	2
14998	<b>_SC_SS_REPL_MAX</b> , <b>_SC_TRACE_EVENT_NAME_MAX</b> , <b>_SC_TRACE_NAME_MAX</b> ,	2
14999	<b>_SC_TRACE_SYS_MAX</b> , and <b>_SC_TRACE_USER_EVENT_MAX</b> to the list of symbolic	2
15000	constants for <i>sysconf()</i> .	2
15001	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/34 is applied, updating the prototype for	2
15002	the <i>symlink()</i> function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.	2
15003	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/35 is applied, adding <b>_PC_2_SYMLINKS</b>	2
15004	to the symbolic constants list for <i>pathconf()</i> . This corresponds to the definition of	2
15005	<b>POSIX2_SYMLINKS</b> in the Shell and Utilities volume of IEEE Std 1003.1-2001.	2

15006 **NAME**

15007       utime.h — access and modification times structure

15008 **SYNOPSIS**

15009       #include <utime.h>

15010 **DESCRIPTION**

15011       The <utime.h> header shall declare the structure **utimbuf**, which shall include the following members:

15013       time\_t     actime     Access time.  
15014       time\_t     modtime   Modification time.

15015       The times shall be measured in seconds since the Epoch.

15016       The type **time\_t** shall be defined as described in <sys/types.h>.

15017       The following shall be declared as a function and may also be defined as a macro. A function  
15018       prototype shall be provided.

15019       int utime(const char \*, const struct utimbuf \*);

15020 **APPLICATION USAGE**

15021       None.

15022 **RATIONALE**

15023       None.

15024 **FUTURE DIRECTIONS**

15025       None.

15026 **SEE ALSO**

15027       <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *utime()*

15028 **CHANGE HISTORY**

15029       First released in Issue 3.

15030 **Issue 6**

15031       The following new requirements on POSIX implementations derive from alignment with the  
15032       Single UNIX Specification:

- 15033       • The **time\_t** type is defined.

## 15034 NAME

15035 utmpx.h — user accounting database definitions

## 15036 SYNOPSIS

15037 XSI `#include <utmpx.h>`

15038

## 15039 DESCRIPTION

15040 The <utmpx.h> header shall define the **utmpx** structure that shall include at least the following  
15041 members:

15042	char	ut_user[ ]	User login name.
15043	char	ut_id[ ]	Unspecified initialization process identifier.
15044	char	ut_line[ ]	Device name.
15045	pid_t	ut_pid	Process ID.
15046	short	ut_type	Type of entry.
15047	struct timeval	ut_tv	Time entry was made.

15048 The **pid\_t** type shall be defined through **typedef** as described in <sys/types.h>.15049 The **timeval** structure shall be defined as described in <sys/time.h>.

15050 Inclusion of the &lt;utmpx.h&gt; header may also make visible all symbols from &lt;sys/time.h&gt;.

15051 The following symbolic constants shall be defined as possible values for the *ut\_type* member of  
15052 the **utmpx** structure:

15053	EMPTY	No valid user accounting information.
15054	BOOT_TIME	Identifies time of system boot.
15055	OLD_TIME	Identifies time when system clock changed.
15056	NEW_TIME	Identifies time after system clock changed.
15057	USER_PROCESS	Identifies a process.
15058	INIT_PROCESS	Identifies a process spawned by the init process.
15059	LOGIN_PROCESS	Identifies the session leader of a logged-in user.
15060	DEAD_PROCESS	Identifies a session leader who has exited.

15061 The following shall be declared as functions and may also be defined as macros. Function  
15062 prototypes shall be provided.

```

15063 void          endutxent(void);
15064 struct utmpx *getutxent(void);
15065 struct utmpx *getutxid(const struct utmpx *);
15066 struct utmpx *getutxline(const struct utmpx *);
15067 struct utmpx *pututxline(const struct utmpx *);
15068 void          setutxent(void);

```

15069 **APPLICATION USAGE**

15070 None.

15071 **RATIONALE**

15072 None.

15073 **FUTURE DIRECTIONS**

15074 None.

15075 **SEE ALSO**

15076 <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *endutxent()*

15077 **CHANGE HISTORY**

15078 First released in Issue 4, Version 2.

## 15079 NAME

15080       wchar.h — wide-character handling

## 15081 SYNOPSIS

15082       #include &lt;wchar.h&gt;

## 15083 DESCRIPTION

15084 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 15085       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 15086       IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
 15087       symbols in this header.

15088       The &lt;wchar.h&gt; header shall define the following types:

15089       **wchar\_t**       As described in <stddef.h>.15090       **wint\_t**       An integer type capable of storing any valid value of **wchar\_t** or WEOF.

15091 XSI       **wctype\_t**   A scalar type of a data object that can hold values which represent locale-  
 15092       specific character classification.

15093       **mbstate\_t**       An object type other than an array type that can hold the conversion state  
 15094       information necessary to convert between sequences of (possibly multi-byte)  
 15095 XSI       characters and wide characters. If a codeset is being used such that an  
 15096       **mbstate\_t** needs to preserve more than 2 levels of reserved state, the results  
 15097       are unspecified.

15098 XSI       **FILE**       As described in <stdio.h>.15099       **size\_t**       As described in <stddef.h>.15100 XSI       **va\_list**       As described in <stdarg.h>.

15101       The implementation shall support one or more programming environments in which the width  
 15102       of **wint\_t** is no greater than the width of type **long**. The names of these programming  
 15103       environments can be obtained using the *confstr()* function or the *getconf* utility.

15104       The following shall be declared as functions and may also be defined as macros. Function  
 15105       prototypes shall be provided.

```

15106       wint_t       btowc(int);
15107       wint_t       fgetwc(FILE *);
15108       wchar_t      *fgetws(wchar_t *restrict, int, FILE *restrict);
15109       wint_t       fputwc(wchar_t, FILE *);
15110       int       fputws(const wchar_t *restrict, FILE *restrict);
15111       int       fwide(FILE *, int);
15112       int       fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15113       int       fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15114       wint_t       getwc(FILE *);
15115       wint_t       getwchar(void);
15116 XSI       int       iswalnum(wint_t);
15117       int       iswalpha(wint_t);
15118       int       iswcntrl(wint_t);
15119       int       iswctype(wint_t, wctype_t);
15120       int       iswdigit(wint_t);
15121       int       iswgraph(wint_t);
15122       int       iswlower(wint_t);
15123       int       iswprint(wint_t);
15124       int       iswpunct(wint_t);
  
```



15125	int	iswspace(wint_t);
15126	int	iswupper(wint_t);
15127	int	iswxdigit(wint_t);
15128	size_t	mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15129	size_t	mbrtowc(wchar_t *restrict, const char *restrict, size_t,
15130		mbstate_t *restrict);
15131	int	mbsinit(const mbstate_t *);
15132	size_t	mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
15133		mbstate_t *restrict);
15134	wint_t	putwc(wchar_t, FILE *);
15135	wint_t	putwchar(wchar_t);
15136	int	swprintf(wchar_t *restrict, size_t,
15137		const wchar_t *restrict, ...);
15138	int	swscanf(const wchar_t *restrict,
15139		const wchar_t *restrict, ...);
15140 XSI	wint_t	towlower(wint_t);
15141	wint_t	towupper(wint_t);
15142	wint_t	ungetwc(wint_t, FILE *);
15143	int	vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15144	int	vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15145	int	vwprintf(const wchar_t *restrict, va_list);
15146	int	vswprintf(wchar_t *restrict, size_t,
15147		const wchar_t *restrict, va_list);
15148	int	vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15149		va_list);
15150	int	vwscanf(const wchar_t *restrict, va_list);
15151	size_t	wcrtomb(char *restrict, wchar_t, mbstate_t *restrict);
15152	wchar_t	*wcscat(wchar_t *restrict, const wchar_t *restrict);
15153	wchar_t	*wcschr(const wchar_t *, wchar_t);
15154	int	wcscmp(const wchar_t *, const wchar_t *);
15155	int	wcscoll(const wchar_t *, const wchar_t *);
15156	wchar_t	*wcscpy(wchar_t *restrict, const wchar_t *restrict);
15157	size_t	wcscspn(const wchar_t *, const wchar_t *);
15158	size_t	wcsftime(wchar_t *restrict, size_t,
15159		const wchar_t *restrict, const struct tm *restrict);
15160	size_t	wcslen(const wchar_t *);
15161	wchar_t	*wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15162	int	wcsncmp(const wchar_t *, const wchar_t *, size_t);
15163	wchar_t	*wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15164	wchar_t	*wcspbrk(const wchar_t *, const wchar_t *);
15165	wchar_t	*wcsrchr(const wchar_t *, wchar_t);
15166	size_t	wcsrtombs(char *restrict, const wchar_t **restrict,
15167		size_t, mbstate_t *restrict);
15168	size_t	wcsspn(const wchar_t *, const wchar_t *);
15169	wchar_t	*wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15170	double	wctod(const wchar_t *restrict, wchar_t **restrict);
15171	float	wctof(const wchar_t *restrict, wchar_t **restrict);
15172	wchar_t	*wcstok(wchar_t *restrict, const wchar_t *restrict,
15173		wchar_t **restrict);
15174	long	wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15175	long double	wcstold(const wchar_t *restrict, wchar_t **restrict);
15176	long long	wcstoll(const wchar_t *restrict, wchar_t **restrict, int);

```

15177 unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15178 unsigned long long
15179         wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15180 XSI wchar_t *wcswcs(const wchar_t *, const wchar_t *);
15181 int wcswidth(const wchar_t *, size_t);
15182 size_t wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15183 int wctob(wint_t);
15184 XSI wctype_t wctype(const char *);
15185 int wcwidth(wchar_t);
15186 wchar_t *wmemchr(const wchar_t *, wchar_t, size_t);
15187 int wmemcmp(const wchar_t *, const wchar_t *, size_t);
15188 wchar_t *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15189 wchar_t *wmemmove(wchar_t *, const wchar_t *, size_t);
15190 wchar_t *wmemset(wchar_t *, wchar_t, size_t);
15191 int wprintf(const wchar_t *restrict, ...);
15192 int wscanf(const wchar_t *restrict, ...);

```

15193 The <wchar.h> header shall define the following macros:

15194 WCHAR\_MAX The maximum value representable by an object of type **wchar\_t**.

15195 WCHAR\_MIN The minimum value representable by an object of type **wchar\_t**.

15196 WEOF Constant expression of type **wint\_t** that is returned by several WP functions

15197 to indicate end-of-file.

15198 NULL As described in <stddef.h>.

15199 The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are

15200 described in the header <time.h>.

15201 CX Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,  
15202 <string.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, and <time.h>.

## 15203 APPLICATION USAGE

15204 The *iswblank()* function was a late addition to the ISO C standard and was introduced at the 1  
15205 same time as the ISO C standard introduced <wctype.h>, which contains all of the *isw\*()* 1  
15206 functions. The Open Group Base Specifications had previously aligned with the MSE working 1  
15207 draft and had introduced the rest of the *isw\*()* functions into <wchar.h>. For backwards- 1  
15208 compatibility, the original set of *isw\*()* functions, without *iswblank()*, are permitted (as an XSI 1  
15209 extension) in <wchar.h>. For maximum portability, applications should include <wctype.h> in 1  
15210 order to obtain declarations for the *isw\*()* functions. 1

## 15211 RATIONALE

15212 In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their  
15213 presence here is thus an extension.

## 15214 FUTURE DIRECTIONS

15215 None.

## 15216 SEE ALSO

15217 <ctype.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, <wctype.h>, the  
15218 System Interfaces volume of IEEE Std 1003.1-2001, *btowc()*, *confstr()*, *fgetc()*, *fgetws()*, *fputc()*,  
15219 *fputws()*, *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpalpha()*, *iswcntrl()*,  
15220 *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*,  
15221 *iswxdigit()*, *iswctype()*, *mbsinit()*, *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *putwc()*, *putwchar()*,  
15222 *swprintf()*, *swscanf()*, *towlower()*, *towupper()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vswprintf()*,  
15223 *vswscanf()*, *vwscanf()*, *wcrtomb()*, *wcsrtombs()*, *wcscat()*, *wcschr()*, *wscmp()*, *wscoll()*, *wscpy()*,

15224 *wscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsspn()*,  
 15225 *wcsstr()*, *wcstod()*, *wcstof()*, *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstoul()*, *wcstoull()*, *wcswcs()*,  
 15226 *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*,  
 15227 *wmemmove()*, *wmemset()*, *wprintf()*, *wscanf()*, the Shell and Utilities volume of  
 15228 IEEE Std 1003.1-2001, *getconf*

## 15229 CHANGE HISTORY

15230 First released in Issue 4.

## 15231 Issue 5

15232 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

## 15233 Issue 6

15234 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and  
 15235 *wcwidth()* are marked as extensions.

15236 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()*  
 15237 function.

15238 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15239 • Various function prototypes are updated to add the **restrict** keyword.
- 15240 • The functions *vfwscanf()*, *vswscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are added.

15241 The type **wctype\_t**, the *isw\*()*, *to\*()*, and *wctype()* functions are marked as XSI extensions.

15242 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/26 is applied, adding the APPLICATION 1  
 15243 USAGE section. 1

## 15244 NAME

15245 wctype.h — wide-character classification and mapping utilities

## 15246 SYNOPSIS

15247 #include &lt;wctype.h&gt;

## 15248 DESCRIPTION

15249 CX Some of the functionality described on this reference page extends the ISO C standard.  
 15250 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 15251 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these  
 15252 symbols in this header.

15253 The &lt;wctype.h&gt; header shall define the following types:

15254 **wint\_t** As described in <wchar.h>.

15255 **wctrans\_t** A scalar type that can hold values which represent locale-specific character  
 15256 mappings.

15257 **wctype\_t** As described in <wchar.h>.

15258 The following shall be declared as functions and may also be defined as macros. Function  
 15259 prototypes shall be provided.

```
15260 int      iswalnum(wint_t);
15261 int      iswalpha(wint_t);
15262 int      iswblank(wint_t);
15263 int      iswcntrl(wint_t);
15264 int      iswdigit(wint_t);
15265 int      iswgraph(wint_t);
15266 int      iswlower(wint_t);
15267 int      iswprint(wint_t);
15268 int      iswpunct(wint_t);
15269 int      iswspace(wint_t);
15270 int      iswupper(wint_t);
15271 int      iswxdigit(wint_t);
15272 int      iswctype(wint_t, wctype_t);
15273 wint_t   towctrans(wint_t, wctrans_t);
15274 wint_t   tolower(wint_t);
15275 wint_t   towupper(wint_t);
15276 wctrans_t wctrans(const char *);
15277 wctype_t  wctype(const char *);
```

15278 The &lt;wctype.h&gt; header shall define the following macro name:

15279 **WEOF** Constant expression of type **wint\_t** that is returned by several MSE functions  
 15280 to indicate end-of-file.

15281 For all functions described in this header that accept an argument of type **wint\_t**, the value is  
 15282 representable as a **wchar\_t** or equals the value of **WEOF**. If this argument has any other value,  
 15283 the behavior is undefined.

15284 The behavior of these functions shall be affected by the *LC\_CTYPE* category of the current locale.

15285 CX Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>,  
 15286 <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, and <wchar.h>.

15287 **APPLICATION USAGE**

15288           None.

15289 **RATIONALE**

15290           None.

15291 **FUTURE DIRECTIONS**

15292           None.

15293 **SEE ALSO**

15294           <ctype.h>, <locale.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>,  
 15295           <wchar.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *iswalnum()*, *iswalpha()*,  
 15296           *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 15297           *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *towctrans()*, *towlower()*, *towupper()*, *wctrans()*,  
 15298           *wctype()*

15299 **CHANGE HISTORY**

15300           First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15301 **Issue 6**

15302           The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

## 15303 NAME

15304 wordexp.h — word-expansion types

## 15305 SYNOPSIS

15306 #include &lt;wordexp.h&gt;

## 15307 DESCRIPTION

15308 The <wordexp.h> header shall define the structures and symbolic constants used by the  
15309 *wordexp()* and *wordfree()* functions.15310 The structure type **wordexp\_t** shall contain at least the following members:

15311 `size_t we_wordc` Count of words matched by *words*.  
 15312 `char **we_wordv` Pointer to list of expanded words.  
 15313 `size_t we_offs` Slots to reserve at the beginning of *we\_wordv*.

15314 The *flags* argument to the *wordexp()* function shall be the bitwise-inclusive OR of the following  
15315 flags:

15316 **WRDE\_APPEND** Append words to those previously generated.  
 15317 **WRDE\_DOOFFS** Number of null pointers to prepend to *we\_wordv*.  
 15318 **WRDE\_NOCMD** Fail if command substitution is requested.  
 15319 **WRDE\_REUSE** The *pwordexp* argument was passed to a previous successful call to  
 15320 *wordexp()*, and has not been passed to *wordfree()*. The result is the same  
 15321 as if the application had called *wordfree()* and then called *wordexp()*  
 15322 without **WRDE\_REUSE**.

15323 **WRDE\_SHOWERR** Do not redirect *stderr* to */dev/null*.15324 **WRDE\_UNDEF** Report error on an attempt to expand an undefined shell variable.

15325 The following constants shall be defined as error return values:

15326 **WRDE\_BADCHAR** One of the unquoted characters—<newline>, ' | ', '&', ';', '<', '>',  
 15327 ' ( ', ' ) ', ' { ', ' } '—appears in *words* in an inappropriate context.  
 15328 **WRDE\_BADVAL** Reference to undefined shell variable when **WRDE\_UNDEF** is set in *flags*.  
 15329 **WRDE\_CMDSUB** Command substitution requested when **WRDE\_NOCMD** was set in *flags*.  
 15330 **WRDE\_NOSPACE** Attempt to allocate memory failed.

15331 OB XSI **WRDE\_NOSYS** Reserved.15332 **WRDE\_SYNTAX** Shell syntax error, such as unbalanced parentheses or unterminated  
15333 string.

15334 The &lt;wordexp.h&gt; header shall define the following type:

15335 XSI **size\_t** As described in <stddef.h>.15336 The following shall be declared as functions and may also be defined as macros. Function  
15337 prototypes shall be provided.

15338 `int wordexp(const char *restrict, wordexp_t *restrict, int);`  
 15339 `void wordfree(wordexp_t *);`

15340 The implementation may define additional macros or constants using names beginning with  
15341 **WRDE\_**.

15342 **APPLICATION USAGE**

15343       None.

15344 **RATIONALE**

15345       None.

15346 **FUTURE DIRECTIONS**

15347       None.

15348 **SEE ALSO**

15349       <stddef.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *wordexp()*, the Shell and  
15350       Utilities volume of IEEE Std 1003.1-2001

15351 **CHANGE HISTORY**

15352       First released in Issue 4. Derived from the ISO POSIX-2 standard.

15353 **Issue 6**

15354       The **restrict** keyword is added to the prototype for *wordexp()*.

15355       The WRDE\_NOSYS constant is marked obsolescent.





# Index

(time) resolution .....	79	<poll.h>.....	288
/ .....	185	<pthread.h> .....	290
/dev .....	185	<pwd.h> .....	295
/dev/console .....	185	<regex.h>.....	296
/dev/null.....	185	<sched.h> .....	298
/dev/tty .....	185	<search.h>.....	300
/tmp .....	185	<semaphore.h> .....	302
<aio.h> .....	206	<setjmp.h> .....	303
<alert>.....	36	<signal.h>.....	304
<arpa/inet.h> .....	208	<space> .....	84
<assert.h> .....	209	<spawn.h> .....	312
<backspace> .....	40	<stdarg.h> .....	314
<blank> .....	45	<stdbool.h>.....	316
<carriage-return>.....	47	<stddef.h> .....	317
<complex.h> .....	210	<stdint.h> .....	318
<cpio.h> .....	213	<stdio.h>.....	325
<ctype.h>.....	214	<stdlib.h> .....	329
<dirent.h>.....	216	<string.h> .....	333
<dlfcn.h> .....	218	<strings.h> .....	335
<errno.h>.....	219	<stropts.h> .....	336
<fcntl.h> .....	223	<sys/dir.h> .....	216
<fenv.h>.....	226	<sys/ipc.h>.....	341
<float.h> .....	229	<sys/mman.h>.....	343
<fmtmsg.h> .....	233	<sys/msg.h>.....	346
<fnmatch.h> .....	235	<sys/resource.h>.....	347
<form-feed>.....	60	<sys/select.h> .....	349
<ftw.h> .....	236	<sys/sem.h> .....	351
<glob.h>.....	238	<sys/shm.h>.....	353
<grp.h> .....	240	<sys/socket.h> .....	355
<iconv.h>.....	241	<sys/stat.h> .....	360
<inttypes.h>.....	242	<sys/statvfs.h> .....	365
<iso646.h> .....	244	<sys/time.h> .....	367
<langinfo.h> .....	245	<sys/timeb.h>.....	369
<libgen.h> .....	248	<sys/times.h> .....	370
<limits.h> .....	249	<sys/types.h> .....	371
<locale.h> .....	264	<sys/uio.h>.....	374
<math.h> .....	266	<sys/un.h>.....	375
<monetary.h> .....	273	<sys/utsname.h>.....	376
<mqueue.h>.....	274	<sys/wait.h> .....	377
<ndbm.h>.....	276	<syslog.h> .....	379
<net/if.h>.....	277	<tab> .....	89
<netdb.h> .....	278	<tar.h>.....	381
<netinet/in.h>.....	282	<termios.h>.....	383
<netinet/tcp.h>.....	286	<tgmath.h> .....	389
<newline>.....	68	<time.h> .....	393
<nl_types.h> .....	287	<trace.h>.....	397

<ucontext.h>.....	401	_POSIX minimum values	
<ulimit.h>.....	402	in <limits.h> .....	255
<unistd.h>.....	403	_POSIX2_BC_BASE_MAX .....	253, 257
<utime.h>.....	423	_POSIX2_BC_DIM_MAX .....	254, 257
<utmpx.h>.....	424	_POSIX2_BC_SCALE_MAX .....	254, 258
<vertical-tab> .....	94	_POSIX2_BC_STRING_MAX .....	254, 258
<wchar.h>.....	426	_POSIX2_CHARCLASS_NAME_MAX.....	254, 258
<wctype.h>.....	430	_POSIX2_CHAR_TERM .....	408
<wordexp.h>.....	432	_POSIX2_COLL_WEIGHTS_MAX .....	254, 258
±0.....	96	_POSIX2_C_BIND .....	408
_Complex_I.....	210	_POSIX2_C_DEV .....	408
_CS_PATH .....	410	_POSIX2_EXPR_NEST_MAX.....	254, 258
_CS_POSIX_V6_ILP32_OFF32_CFLAGS.....	410	_POSIX2_FORT_DEV .....	408
_CS_POSIX_V6_ILP32_OFF32_LDFLAGS .....	411	_POSIX2_FORT_RUN .....	408
_CS_POSIX_V6_ILP32_OFF32_LIBS.....	411	_POSIX2_LINE_MAX .....	254, 258, 260
_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS.....	411	_POSIX2_LOCALEDEF.....	408
_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS .....	411	_POSIX2_PBS .....	408
_CS_POSIX_V6_ILP32_OFFBIG_LIBS.....	411	_POSIX2_PBS_ACCOUNTING .....	408
_CS_POSIX_V6_LP64_OFF64_CFLAGS .....	411	_POSIX2_PBS_CHECKPOINT .....	408
_CS_POSIX_V6_LP64_OFF64_LDFLAGS.....	411	_POSIX2_PBS_LOCATE.....	408
_CS_POSIX_V6_LP64_OFF64_LIBS .....	411	_POSIX2_PBS_MESSAGE.....	408
_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS.....	411	_POSIX2_PBS_TRACK.....	409
_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS...411		_POSIX2_RE_DUP_MAX.....	251, 254, 258
_CS_POSIX_V6_LPBIG_OFFBIG_LIBS .....	412	_POSIX2_SW_DEV .....	409
_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS....		_POSIX2_UPE .....	409
.....	412	_POSIX2_VERSION .....	403
_CS_XBS5_ILP32_OFF32_CFLAGS.....	412	_POSIX_ADVISORY_INFO .....	19, 25-26, 404
_CS_XBS5_ILP32_OFF32_LDFLAGS .....	412	_POSIX_AIO_LISTIO_MAX .....	250, 255
_CS_XBS5_ILP32_OFF32_LIBS.....	412	_POSIX_AIO_MAX .....	250, 255
_CS_XBS5_ILP32_OFF32_LINTFLAGS.....	412	_POSIX_ARG_MAX .....	250, 255
_CS_XBS5_ILP32_OFFBIG_LDFLAGS .....	412	_POSIX_ASYNCHRONOUS_IO .....	19, 24-25, 404
_CS_XBS5_ILP32_OFFBIG_LIBS.....	412	_POSIX_ASYNC_IO .....	410
_CS_XBS5_ILP32_OFFBIG_LINTFLAGS.....	412	_POSIX_BARRIERS .....	19, 27, 404
_CS_XBS5_LP64_OFF64_CFLAGS .....	412	_POSIX_CHILD_MAX.....	250, 255
_CS_XBS5_LP64_OFF64_LDFLAGS .....	412	_POSIX_CHOWN_RESTRICTED .....	18, 404
_CS_XBS5_LP64_OFF64_LIBS .....	412	_POSIX_CLOCKRES_MIN.....	254
_CS_XBS5_LP64_OFF64_LINTFLAGS .....	412	_POSIX_CLOCK_SELECTION .....	19, 25-26, 404
_CS_XBS5_LPBIG_OFFBIG_CFLAGS .....	412	_POSIX_CPUTIME.....	19, 25-26, 404
_CS_XBS5_LPBIG_OFFBIG_LDFLAGS.....	412	_POSIX_DELAYTIMER_MAX .....	250, 255
_CS_XBS5_LPBIG_OFFBIG_LIBS .....	412	_POSIX_FSYNC.....	19, 21, 24-25, 404
_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS .....	412	_POSIX_HOST_NAME_MAX.....	250, 255
_Imaginary_I .....	210	_POSIX_IPV6.....	19
_IOFBF .....	325	_POSIX_JOB_CONTROL .....	18, 404
_IOLBF .....	325	_POSIX_LINK_MAX.....	252, 255
_IONBF .....	325	_POSIX_LOGIN_NAME_MAX .....	250, 255
_MIN .....	249	_POSIX_MAPPED_FILES .....	19, 21, 24, 404
_PC constants		_POSIX_MAPPED_FILES, .....	25
defined in <unistd.h> .....	413	_POSIX_MAX_CANON .....	252, 255
_POSIX .....	249	_POSIX_MAX_INPUT .....	253, 255
_POSIX maximum values		_POSIX_MEMLOCK.....	19, 24-25, 404
in <limits.h> .....	254	_POSIX_MEMLOCK_RANGE .....	19, 24-25, 404

<code>_POSIX_MEMORY_PROTECTION</code> .....	19, 21	<code>_POSIX_THREAD_PROCESS_SHARED</code> .....	19
.....	24-25, 405	.....	21, 407
<code>_POSIX_MESSAGE_PASSING</code> .....	19, 24-25, 405	<code>_POSIX_THREAD_SAFE_FUNCTIONS</code> .....	19
<code>_POSIX_MONOTONIC_CLOCK</code> .....	19, 25-26, 405	.....	21, 27, 407
<code>_POSIX_MQ_OPEN_MAX</code> .....	250, 255	<code>_POSIX_THREAD_SPORADIC_SERVER</code> .....	19
<code>_POSIX_MQ_PRIO_MAX</code> .....	250, 255	.....	27, 407
<code>_POSIX_NAME_MAX</code> .....	253, 256	<code>_POSIX_THREAD_THREADS_MAX</code> .....	251, 257
<code>_POSIX_NGROUPS_MAX</code> .....	254, 256	<code>_POSIX_TIMEOUTS</code> .....	20, 25-26, 407
<code>_POSIX_NO_TRUNC</code> .....	18, 100, 405	<code>_POSIX_TIMERS</code> .....	20, 24-27, 407
<code>_POSIX_OPEN_MAX</code> .....	250, 256	<code>_POSIX_TIMER_MAX</code> .....	252, 257
<code>_POSIX_PATH_MAX</code> .....	253, 256, 375	<code>_POSIX_TRACE</code> .....	20, 27, 407
<code>_POSIX_PIPE_BUF</code> .....	253, 256	<code>_POSIX_TRACE_EVENT_FILTER</code> .....	20, 27, 407
<code>_POSIX_PRIORITIZED_IO</code> .....	19, 24-25, 405	<code>_POSIX_TRACE_EVENT_NAME_MAX</code> .....	252, 257
<code>_POSIX_PRIORITY_SCHEDULING</code> .....	19, 24-26, 405	<code>_POSIX_TRACE_INHERIT</code> .....	20, 27, 407
<code>_POSIX_PRIO_IO</code> .....	410	<code>_POSIX_TRACE_LOG</code> .....	20, 27, 407
<code>_POSIX_RAW_SOCKETS</code> .....	19, 405	<code>_POSIX_TRACE_NAME_MAX</code> .....	252, 257
<code>_POSIX_READER_WRITER_LOCKS</code> .....	405	<code>_POSIX_TRACE_SYS_MAX</code> .....	252, 257
<code>_POSIX_REALTIME_SIGNALS</code> .....	19, 24-25, 405	<code>_POSIX_TRACE_USER_EVENT_MAX</code> .....	252, 257
<code>_POSIX_REGEX</code> .....	405	<code>_POSIX_TTY_NAME_MAX</code> .....	252, 257
<code>_POSIX_RE_DUP_MAX</code> .....	256	<code>_POSIX_TYPED_MEMORY_OBJECTS</code> .....	20
<code>_POSIX_RTSIG_MAX</code> .....	251, 256	.....	25-26, 407
<code>_POSIX_SAVED_IDS</code> .....	18, 405	<code>_POSIX_TZNAME_MAX</code> .....	252, 257
<code>_POSIX_SEMAPHORES</code> .....	19, 24-25, 405	<code>_POSIX_VDISABLE</code> .....	18, 408
<code>_POSIX_SEM_NSEMS_MAX</code> .....	251, 256	<code>_POSIX_VERSION</code> .....	18, 403
<code>_POSIX_SEM_VALUE_MAX</code> .....	251, 256	<code>_SC constants</code>	
<code>_POSIX_SHARED_MEMORY_OBJECTS</code> .....	19	defined in <code>&lt;unistd.h&gt;</code> .....	<b>413</b>
.....	24-25, 406	<code>_XBS5_ILP32_OFF32</code> .....	409
<code>_POSIX_SHELL</code> .....	406	<code>_XBS5_ILP32_OFFBIG</code> .....	409
<code>_POSIX_SIGQUEUE_MAX</code> .....	251, 256	<code>_XBS5_LP64_OFF64</code> .....	409
<code>_POSIX_SPAWN</code> .....	19, 25-26, 406	<code>_XBS5_LPBIG_OFFBIG</code> .....	409
<code>_POSIX_SPIN_LOCKS</code> .....	19, 27, 406	<code>_XOPEN_CRYPT</code> .....	20, 24, 409
<code>_POSIX_SPARADIC_SERVER</code> .....	19, 25-26, 406	<code>_XOPEN_ENH_I18N</code> .....	409
<code>_POSIX_SSIZE_MAX</code> .....	256, 259	<code>_XOPEN_IOV_MAX</code> .....	250, 258
<code>_POSIX_SS_REPL_MAX</code> .....	251, 256	<code>_XOPEN_LEGACY</code> .....	20, 28, 409
<code>_POSIX_STREAM_MAX</code> .....	251, 256	<code>_XOPEN_NAME_MAX</code> .....	253, 258
<code>_POSIX_SYMLINK_MAX</code> .....	253, 256	<code>_XOPEN_PATH_MAX</code> .....	253, 258
<code>_POSIX_SYMLOOP_MAX</code> .....	251, 257	<code>_XOPEN_REALTIME</code> .....	20, 24-25, 409
<code>_POSIX_SYNCHRONIZED_IO</code> .....	19, 24-25, 406	<code>_XOPEN_REALTIME_THREADS</code> .....	20, 26, 410
<code>_POSIX_SYNC_IO</code> .....	410	<code>_XOPEN_SHM</code> .....	410
<code>_POSIX_THREADS</code> .....	19, 21, 27, 407	<code>_XOPEN_STREAMS</code> .....	28, 410
<code>_POSIX_THREAD_ATTR_STACKADDR</code> .....	19	<code>_XOPEN_UNIX</code> .....	20-21, 410
.....	21, 406	<code>_XOPEN_VERSION</code> .....	403
<code>_POSIX_THREAD_ATTR_STACKSIZE</code> .....	19, 21, 406	<code>ABDAY</code> .....	246
<code>_POSIX_THREAD_CPUTIME</code> .....	19, 27, 406	<code>ABMON</code> .....	246
<code>_POSIX_THREAD_DESTRUCTOR</code> .....		abortive release.....	35
<code>ITERATIONS</code> .....	251, 257	absolute pathname.....	35, 100
<code>_POSIX_THREAD_KEYS_MAX</code> .....	251, 257	access mode.....	35
<code>_POSIX_THREAD_PRIORITY_SCHEDULING</code> .....		additional file access control mechanism.....	35
.....	19, 26-27, 406	address space.....	35
<code>_POSIX_THREAD_PRIO_INHERIT</code> .....	19, 26, 406	<code>ADV</code> .....	<b>6</b>
<code>_POSIX_THREAD_PRIO_PROTECT</code> .....	19, 26, 406	advanced realtime.....	25

ADVANCED REALTIME.....	312	asynchronously-generated signal .....	39
advanced realtime threads .....	26	ATEXIT_MAX .....	250
advisory information.....	35	attribute selection.....	386
affirmative response.....	36	authentication.....	39
AF_INET.....	357	authorization .....	39
AF_INET6 .....	357	background job .....	39
AF_UNIX.....	357	background process .....	39
AF_UNSPEC.....	357	background process group.....	39
AIO .....	6	backquote .....	40
AIO_ALLDONE .....	206	BACKREF.....	179
AIO_CANCELED.....	206	backslash .....	40
AIO_LISTIO_MAX.....	249	backspace character .....	40
AIO_MAX .....	250	bandinfo.....	336
AIO_NOTCANCELED .....	206	BAR.....	7
AIO_PRIO_DELTA_MAX.....	250	barrier.....	40
AI_ADDRCONFIG .....	279	base character .....	40
AI_ALL .....	279	basename.....	40
AI_CANONNAME.....	279	basic regular expression.....	40, 171
AI_NUMERICHOST.....	279	batch access list .....	40
AI_NUMERICSERV.....	279	batch administrator.....	41
AI_PASSIVE.....	279	batch client .....	41
AI_V4MAPPED.....	279	batch destination .....	41
alert.....	36	batch destination identifier.....	41
alert character.....	36	batch directive.....	41
alias name.....	36	batch job.....	41
alignment.....	36	batch job attribute.....	42
alternate file access control mechanism.....	36	batch job identifier.....	42
alternate signal stack.....	37	batch job name .....	42
ALT_DIGITS.....	246	batch job owner.....	42
AM_STR .....	246	batch job priority .....	42
anchoring.....	175	batch job state.....	42
ancillary data .....	37	batch name service .....	42
angle brackets.....	37	batch name space.....	42
ANYMARK.....	339	batch node.....	43
API.....	37	batch operator .....	43
application .....	37	batch queue.....	43
application address.....	37	batch queue attribute .....	43
application conformance .....	31	batch queue position.....	43
application program interface .....	37	batch queue priority.....	43
appropriate privileges .....	37	batch rerunability .....	43
AREGTYPE.....	381	batch restart .....	44
argument .....	38	batch server.....	44
ARG_MAX.....	250	batch server name.....	44
arm (a timer) .....	38	batch service .....	44
asterisk.....	38	batch service request.....	44
async-cancel-safe function.....	38	batch submission .....	44
async-signal-safe function .....	38	batch system .....	44
asynchronous events .....	38	batch target user .....	45
asynchronous I/O completion .....	39	batch user .....	45
asynchronous I/O operation .....	39	baud rate selection .....	385
asynchronous input and output.....	38	BC_BASE_MAX.....	253

BC_DIM_MAX.....	253	CHILD_MAX.....	250
BC_SCALE_MAX.....	254	CHRTYPE.....	381
BC_STRING_MAX.....	254	circumflex.....	48
BE.....	7	CLD_CONTINUED.....	308
bind.....	45	CLD_DUMPED.....	308
blank character.....	45	CLD_EXITED.....	308
blank line.....	45	CLD_KILLED.....	308
blkcnt_t.....	371	CLD_STOPPED.....	308
blksize_t.....	371	CLD_TRAPPED.....	308
BLKTYPE.....	381	CLOCAL.....	386
block special file.....	46	clock.....	48
block-mode terminal.....	45	clock jump.....	48
blocked process (or thread).....	45	clock tick.....	48
blocking.....	45	clockid_t.....	371
BOOT_TIME.....	424	CLOCKS_PER_SEC.....	371, 393-394
braces.....	46	CLOCK_MONOTONIC.....	394
brackets.....	46	CLOCK_PROCESS_CPUTIME_ID.....	393
BRE (ERE) matching a single character.....	170	CLOCK_REALTIME.....	254, 393
BRE (ERE) matching multiple characters.....	170	clock_t.....	371
BRKINT.....	384	CLOCK_THREAD_CPUTIME_ID.....	393
broadcast.....	46	CMMSG_DATA.....	356
BSD.....	216	CMMSG_FIRSTHDR.....	356
BSDLY.....	385	CMMSG_NXTHDR.....	356
BSn.....	385	coded character set.....	48
BUFSIZ.....	325	codeset.....	49
built-in.....	46	CODESET.....	246
built-in utility.....	46	collating element.....	49
BUS_ADRALN.....	308	collation.....	49
BUS_ADRERR.....	308	collation sequence.....	49
BUS_OBJERR.....	308	COLL_ELEM_MULTI.....	179
byte.....	46	COLL_ELEM_SINGLE.....	179
byte input/output functions.....	47	COLL_WEIGHTS_MAX.....	254
can.....	5	column position.....	49
canonical mode input processing.....	189	COLUMNS.....	165
carriage-return character.....	47	command.....	50
CD.....	7	command language interpreter.....	50
character.....	47	complex.....	210
character array.....	47	composite graphic symbol.....	50
character class.....	47	concurrent execution.....	97
character encoding.....	116	condition variable.....	50
state-dependent.....	120	conformance.....	17, 31
character set.....	47	POSIX.....	17
character special file.....	48	POSIX system interfaces.....	18
character string.....	48	XSI.....	17
CHARCLASS_NAME_MAX.....	254	XSI system interfaces.....	21
charmap.....		conformance document.....	17
description.....	117	conforming application.....	17
CHAR_BIT.....	258-259	conforming implementation options.....	22
CHAR_MAX.....	259	connection.....	50
CHAR_MIN.....	259	connection mode.....	50
child process.....	48	connectionless mode.....	50

control character .....	51	entry .....	300
control modes .....	386	group .....	240
control operator .....	51	lconv .....	264
controlling process .....	51	msqid_ds .....	346
controlling terminal .....	51, 188	stat .....	360
CONTTYTYPE .....	381	tms .....	370
conversion descriptor .....	51	utimbuf .....	423
copy .....	142	data type	
core file .....	51	ACTION .....	300
CPT .....	7	cc_t .....	383
CPU .....	370	DIR .....	216
CPU time .....	51	div_t .....	329
clock .....	52	ENTRY .....	300
timer .....	52	FILE .....	326
CRDLY .....	384	fpos_t .....	326
CREAD .....	386	glob_t .....	238
CRn .....	384	ldiv_t .....	329
CRNCYSTR .....	246	lldiv_t .....	329
CS .....	7	mbstate_t .....	426
CSIZE .....	386	msglen_t .....	346
CSn .....	386	msgqnum_t .....	346
CSTOPB .....	386	nl_catd .....	287
currency_symbol .....	142	nl_item .....	287
current job .....	52	pid_t .....	304
current working directory .....	52, 95	ptrdiff_t .....	317
cursor position .....	52	regex_t .....	296
CX .....	7	regmatch_t .....	296
C_ constants in <cpio.h> .....	<b>213</b>	regoff_t .....	296
C_IRGRP .....	213	shmatt_t .....	353
C_IROTH .....	213	sigset_t .....	304
C_IRUSR .....	213	sig_atomic_t .....	304
C_ISBLK .....	213	size_t .....	317
C_ISCHR .....	213	speed_t .....	383
C_ISCTG .....	213	tflag_t .....	383
C_ISDIR .....	213	VISIT .....	300
C_ISFIFO .....	213	wchar_t .....	317
C_ISGID .....	213	wctrans_t .....	430
C_ISLNK .....	213	wctype_t .....	426
C_ISREG .....	213	wint_t .....	426
C_ISSOCK .....	213	data types	
C_ISUID .....	213	defined in <fenv.h> .....	<b>226</b>
C_ISVTX .....	213	defined in <sys/types.h> .....	371
C_IWGRP .....	213	DATMSK .....	<b>165</b>
C_IWOTH .....	213	DAY .....	246
C_IWUSR .....	213	DBL_ constants	
C_IXGRP .....	213	defined in <float.h> .....	<b>230</b>
C_IXOTH .....	213	DBL_DIG .....	230, 258
C_IXUSR .....	213	DBL_EPSILON .....	231
data segment .....	52	DBL_MANT_DIG .....	230
data structure		DBL_MAX .....	231, 258
dirent .....	216	DBL_MAX_10_EXP .....	231

DBL_MAX_EXP .....	231	EAI_SYSTEM.....	280
DBL_MIN .....	232	EALREADY .....	219
DBL_MIN_10_EXP .....	231	EBADF .....	219
DBL_MIN_EXP .....	231	EBADMSG .....	219
DBM.....	276	EBUSY .....	219
DBM_INSERT .....	276	ECANCELED .....	219
DBM_REPLACE .....	276	ECHILD .....	219
DEAD_PROCESS .....	424	ECHO .....	386
DECIMAL_DIG.....	230	ECHOE .....	386
deferred batch service.....	52	ECHOK.....	386
DELAYTIMER_MAX .....	250	ECHONL.....	386
device .....	52	ECONNABORTED .....	219
output .....	185	ECONNREFUSED.....	219
device ID.....	52	ECONNRESET.....	219
dev_t.....	371	EDEADLK .....	219
DIR.....	216	EDESTADDRREQ .....	219
directory .....	53	EDOM .....	219
directory entry (or link) .....	53	EDQUOT .....	219
directory protection .....	97	EEXIST .....	219
directory stream.....	53	EFAULT .....	219
dirent structure .....	216	EFBIG .....	219
DIRTYPE .....	381	effective group ID.....	54
disarm (a timer).....	53	effective user ID .....	54
display.....	53	EHOSTUNREACH .....	219
display line.....	53	EIDRM .....	219
documentation .....	17	eight-bit transparency.....	54
dollar sign.....	53	EILSEQ.....	219
domain error.....	105	EINPROGRESS .....	220
dot.....	53	EINTR .....	220
dot-dot .....	54	EINVAL .....	220
double-quote.....	54	EIO .....	220
downshifting .....	54	EISCONN.....	220
driver.....	54	EISDIR.....	220
DUP_COUNT.....	179	ELOOP .....	220
D_FMT .....	246	EMFILE.....	220
D_T_FMT .....	246	EMLINK .....	220
E2BIG .....	219	EMPTY.....	424
EACCES.....	219	empty directory.....	54
EADDRINUSE .....	219	empty line.....	55
EADDRNOTAVAIL.....	219	empty string (or null string).....	55
EAFNOSUPPORT .....	219	empty wide-character string.....	55
EAGAIN .....	219	EMSGSIZE .....	220
EAI_AGAIN .....	280	EMULTIHOP .....	220
EAI_BADFLAGS.....	280	ENAMETOOLONG .....	220
EAI_FAIL.....	280	encoding	
EAI_FAMILY .....	280	character .....	116
EAI_MEMORY.....	280	encoding rule.....	55
EAI_NONAME.....	280	encryption .....	24
EAI_OVERFLOW .....	280	ENETDOWN.....	220
EAI_SERVICE.....	280	ENETRESET .....	220
EAI_SOCKTYPE.....	280	ENETUNREACH .....	220

ENFILE .....	220	EWOULDLOCK .....	221
ENOBUFS .....	220	EXDEV .....	221
ENODATA .....	220	executable file .....	56
ENODEV .....	220	execute .....	56
ENOENT .....	220	execution time .....	51, 56
ENOEXEC .....	220	measurement .....	99
ENOLCK .....	220	monitoring .....	56
ENOLINK .....	220	EXIT_FAILURE .....	329
ENOMEM .....	220	EXIT_SUCCESS .....	329
ENOMSG .....	220	expand .....	56
ENOPROTOPT .....	220	EXPR_NEST_MAX .....	254
ENOSPC .....	220	extended regular expression .....	56, 175
ENOSR .....	220	extended security controls .....	57, 97
ENOSTR .....	220	extension	
ENOSYS .....	220	CX .....	7
ENOTCONN .....	220	OH .....	10
ENOTDIR .....	220	XSI .....	14
ENOTEMPTY .....	220	F-LOCK .....	412
ENOTSOCK .....	220	FD .....	7
ENOTSUP .....	221	FD_CLOEXEC .....	223
ENOTTY .....	221	FD_CLR .....	349
entire regular expression .....	55, 169	FD_ISSET .....	349
environment variables		FD_SET .....	349
internationalization .....	162	fd_set .....	349, 367
ENXIO .....	221	FD_SETSIZE .....	349
EOF .....	325	FD_ZERO .....	349
EOPNOTSUPP .....	221	feature test macro .....	57
EOVERFLOW .....	221	fenv_t .....	226
EPERM .....	221	fexcept_t .....	226
EPIPE .....	221	FE_constants	
epoch .....	55	defined in <fenv.h> .....	226
EPROTO .....	221	FE_ALL_EXCEPT .....	226
EPROTONOSUPPORT .....	221	FE_DFL_ENV .....	227
EPROTOTYPE .....	221	FE_DIVBYZERO .....	226
equivalence class .....	55	FE_DOWNWARD .....	226
era .....	55	FE_INEXACT .....	226
ERA .....	246	FE_INVALID .....	226
ERANGE .....	221	FE_OVERFLOW .....	226
ERA_D_FMT .....	246	FE_TONEAREST .....	226
ERA_D_T_FMT .....	246	FE_TOWARDZERO .....	226
ERA_T_FMT .....	246	FE_UNDERFLOW .....	226
EROFS .....	221	FE_UPWARD .....	226
error conditions		FFDLY .....	385
mathematical functions .....	105	FFn .....	385
ESPIPE .....	221	field .....	57
ESRCH .....	221	FIFO .....	57
ESTALE .....	221	FIFO special file .....	57
ETIME .....	221	FIFOTYPE .....	381
ETIMEDOUT .....	221	file .....	57
ETXTBSY .....	221	FILE .....	325, 426
event management .....	56	file access permissions .....	97



file characteristics		
data structure .....	362	
header .....	362	
file description .....	58	
file descriptor .....	58	
file group class .....	58	
file hierarchy .....	98	
file mode .....	58	
file mode bits .....	58	
file offset .....	59	
file other class .....	59	
file owner class .....	59	
file permission bits .....	59	
file serial number .....	59	
file system .....	59	
file times update .....	98	
file type .....	59	
filename .....	58	
filename portability .....	58	
FILENAME_MAX .....	325	
FILESIZEBITS .....	252	
filter .....	60	
FIPS .....	18	
first open (of a file) .....	60	
flow control .....	60	
FLT_ constants		
defined in <float.h> .....	<b>230</b>	
FLT_DIG .....	230, 258	
FLT_EPSILON .....	231	
FLT_EVAL_METHOD .....	229	
FLT_MANT_DIG .....	230	
FLT_MAX .....	231, 258	
FLT_MAX_10_EXP .....	231	
FLT_MAX_EXP .....	231	
FLT_MIN .....	232	
FLT_MIN_10_EXP .....	231	
FLT_MIN_EXP .....	231	
FLT_RADIX .....	230	
FLT_ROUNDS .....	229	
FLUSHR .....	338	
FLUSHRW .....	338	
FLUSHW .....	338	
FMNAMESZ .....	337-338	
FNM_ constants		
in <fnmatch.h> .....	<b>235</b>	
FNM_NOESCAPE .....	235	
FNM_NOMATCH .....	235	
FNM_NOSYS .....	235	
FNM_PATHNAME .....	235	
FNM_PERIOD .....	235	
FOPEN_MAX .....	251, 325	
foreground job .....	60	
foreground process .....	60	
foreground process group .....	60	
foreground process group ID .....	60	
form-feed character .....	60	
format of entries .....	<b>205</b>	
FPE_FLTDIV .....	308	
FPE_FLTINV .....	308	
FPE_FLTOVF .....	308	
FPE_FLTRES .....	308	
FPE_FLTSUB .....	308	
FPE_FLTUND .....	308	
FPE_INTDIV .....	308	
FPE_INTOVF .....	308	
FR .....	7	
frac_digits .....	143	
fsblkcnt_t .....	371	
FSC .....	<b>8</b>	
fsfilcnt_t .....	371	
FTW .....	236	
FTW_ constants		
in <ftw.h> .....	<b>236</b>	
FTW_CHDIR .....	236	
FTW_D .....	236	
FTW_DEPTH .....	236	
FTW_DNR .....	236	
FTW_DP .....	236	
FTW_F .....	236	
FTW_MOUNT .....	236	
FTW_NS .....	236	
FTW_PHYS .....	236	
FTW_SL .....	236	
FTW_SLN .....	236	
F_DUPFD .....	223	
F_GETFD .....	223	
F_GETFL .....	223	
F_GETLK .....	223	
F_GETOWN .....	223	
F_OK .....	410	
F_RDLCK .....	223	
F_SETFD .....	223	
F_SETFL .....	223	
F_SETLK .....	223	
F_SETLKW .....	223	
F_SETOWN .....	223	
F_TEST .....	<b>412</b>	
F_TLOCK .....	<b>412</b>	
F_ULOCK .....	<b>412</b>	
F_UNLCK .....	223	
F_WRLCK .....	223	
GETALL .....	351	

GETNCNT .....	351	ILL_PRVOPC.....	308
GETPID.....	351	ILL_PRIVREG.....	308
GETVAL .....	351	imaginary .....	210
GETZCNT .....	351	implementation-defined.....	5
gid_t.....	371	IN6_IS_ADDR_LINKLOCAL.....	284
GLOB_constants		IN6_IS_ADDR_LOOPBACK.....	284
defined in <glob.h> .....	<b>238</b>	IN6_IS_ADDR_MC_GLOBAL.....	284
GLOB_ABORTED .....	238	IN6_IS_ADDR_MC_LINKLOCAL.....	284
GLOB_APPEND .....	238	IN6_IS_ADDR_MC_NODELOCAL.....	284
GLOB_DOOFFS .....	238	IN6_IS_ADDR_MC_ORGLOCAL.....	284
GLOB_ERR .....	238	IN6_IS_ADDR_MC_SITELOCAL.....	284
GLOB_MARK.....	238	IN6_IS_ADDR_MULTICAST.....	284
GLOB_NOCHECK.....	238	IN6_IS_ADDR_SITELOCAL.....	284
GLOB_NOESCAPE.....	238	IN6_IS_ADDR_UNSPECIFIED .....	284
GLOB_NOMATCH.....	238	IN6_IS_ADDR_V4COMPAT .....	284
GLOB_NOSORT .....	238	IN6_IS_ADDR_V4MAPPED .....	284
GLOB_NOSPACE .....	238	INADDR_ANY .....	283
GLOB_NOSYS.....	238	INADDR_BROADCAST.....	283
grammar		incomplete line.....	62
locale .....	153	INET6_ADDRSTRLEN .....	283
regular expression.....	179	INET_ADDRSTRLEN.....	283
graphic character .....	61	Inf.....	62
group database.....	61	INFINITY .....	267
group ID .....	61	INIT_PROCESS.....	424
group name.....	61	INLCR.....	384
hard limit.....	61	ino_t.....	371
hard link.....	61	INPCK.....	384
headers.....	<b>205</b>	instrumented application.....	62
HOME.....	<b>165</b>	interactive shell.....	62
home directory .....	61	internationalization.....	62
host byte order .....	62, 99	interprocess communication.....	62
HOST_NAME_MAX .....	250	INTMAX_MAX.....	322
HUGE_VAL .....	267	INTMAX_MIN.....	322
HUGE_VALF.....	267	INTN_MAX.....	321
HUGE_VALL.....	267	INTN_MIN .....	321
HUPCL .....	386	INTPTR_MAX.....	321
I .....	210	INTPTR_MIN.....	321
ICANON .....	386	int_curr_symbol.....	142
ICRNL.....	384	INT_FASTN_MAX.....	321
idtype_t.....	<b>377</b>	INT_FASTN_MIN .....	321
id_t.....	371	int_frac_digits.....	143
IEXTEN.....	386	INT_LEASTN_MAX.....	321
IGNBRK.....	384	INT_LEASTN_MIN .....	321
IGNCR .....	384	INT_MAX.....	259
IGNPAR.....	384	INT_MIN.....	260
ILL_BADSTK.....	308	int_n_cs_precedes .....	144
ILL_COPROC.....	308	int_n_sep_by_space .....	144
ILL_ILADR .....	308	int_n_sign_posn.....	144
ILL_ILOPC .....	308	int_p_cs_precedes .....	144
ILL_ILOPN .....	308	int_p_sep_by_space .....	144
ILL_ILLTRP.....	308	int_p_sign_posn.....	144

invalid .....	170	I_LINK .....	337
invariant values .....	260	I_LIST .....	337
invoke .....	62	I_LOOK .....	337
iovec .....	374	I_NREAD .....	337
IOV_MAX .....	250, 374	I_PEEK .....	337
IP6 .....	8	I_PLINK .....	337
IPC .....	341	I_POP .....	337
IPC_constants		I_PUNLINK .....	337
defined in <sys/ipc.h> .....	341	I_PUSH .....	337
IPC_CREAT .....	341	I_RECVFD .....	337
IPC_EXCL .....	341	I_SENDFD .....	337
IPC_NOWAIT .....	341	I_SETCLTIME .....	337
IPC_PRIVATE .....	341	I_SETSIG .....	337
IPC_RMID .....	341	I_SRDOPT .....	337
IPC_SET .....	341	I_STR .....	337
IPC_STAT .....	341	I_SWROPT .....	337
IPPROTO_ICMP .....	283	I_UNLINK .....	337
IPPROTO_IP .....	283	job .....	63
IPPROTO_IPV6 .....	283	job control .....	63
IPPROTO_RAW .....	283	job control job ID .....	63
IPPROTO_TCP .....	283	key_t .....	371
IPPROTO_UDP .....	283	LANG .....	162
IPV6_JOIN_GROUP .....	284	last close (of a file) .....	63
IPV6_LEAVE_GROUP .....	284	LASTMARK .....	339
IPV6_MULTICAST_HOPS .....	284	LC_ALL .....	163, 264
IPV6_MULTICAST_IF .....	284	LC_COLLATE .....	163, 254, 264
IPV6_MULTICAST_LOOP .....	284	description .....	134
IPV6_UNICAST_HOPS .....	284	LC_CTYPE .....	163, 246, 264, 430
IPV6_V6ONLY .....	284	description .....	126
ISIG .....	386	LC_MESSAGES .....	163, 246, 264, 287
ISO C standard .....	210	description .....	152
ISTRIP .....	384	LC_MONETARY .....	163, 246, 264
itimerval .....	367	description .....	142
ITIMER_PROF .....	367	LC_NUMERIC .....	163, 246, 264
ITIMER_REAL .....	367	description .....	145
ITIMER_VIRTUAL .....	367	LC_TIME .....	163, 246, 264
IXANY .....	384	description .....	147
IXOFF .....	384	LDBL_constants	
IXON .....	384	defined in <float.h> .....	230
I_ATMARK .....	337	LDBL_DIG .....	231
I_CANPUT .....	337	LDBL_EPSILON .....	232
I_CKBAND .....	337	LDBL_MANT_DIG .....	230
I_FDINSERT .....	337	LDBL_MAX .....	231
I_FIND .....	337	LDBL_MAX_10_EXP .....	231
I_FLUSH .....	337	LDBL_MAX_EXP .....	231
I_FLUSHBAND .....	337	LDBL_MIN .....	232
I_GETBAND .....	337	LDBL_MIN_10_EXP .....	231
I_GETCLTIME .....	337	LDBL_MIN_EXP .....	231
I_GETSIG .....	337	legacy .....	5, 28
I_GRDOPT .....	337	limit	
I_GWROPT .....	337	numerical .....	258

line .....	63	LOG_UUCP .....	379
line control .....	387	LOG_WARNING .....	380
LINES .....	<b>165</b>	LONG_BIT .....	258-259
LINE_MAX .....	254	LONG_MAX .....	259
linger .....	63	LONG_MIN .....	260
link .....	63	lower multiplexing .....	85
link count .....	64	L_ANCHOR .....	179
LINK_MAX .....	252	L_ctermid .....	325
LIO_NOP .....	206	L_tmpnam .....	325
LIO_NOWAIT .....	206	MAGIC .....	213
LIO_READ .....	206	map .....	64
LIO_WAIT .....	206	MAP_FIXED .....	343
LIO_WRITE .....	206	MAP_PRIVATE .....	343
LLONG_MAX .....	260	MAP_SHARED .....	343
LLONG_MIN .....	260	margin codes	
LNKTYPE .....	381	notation .....	14
local customs .....	64	marked message .....	65
local IPC .....	64	matched .....	65, 169
local modes .....	386	mathematical functions	
locale .....	64, 123	domain error .....	105
grammar .....	153	error conditions .....	105
POSIX .....	124	NaN arguments .....	106
locale definition .....	124	pole error .....	106
localization .....	64	range error .....	106
login .....	64	MAXARGS .....	315
login name .....	64	MAXFLOAT .....	267
LOGIN_NAME_MAX .....	250	maximum values .....	254
LOGIN_PROCESS .....	424	MAX_CANON .....	252
LOGNAME .....	<b>166</b>	MAX_INPUT .....	252
LOG_ALERT .....	380	may .....	5
LOG_AUTH .....	379	MB_CUR_MAX .....	329
LOG_CONS .....	379	MB_LEN_MAX .....	258-259
LOG_CRIT .....	380	MC1 .....	<b>8</b>
LOG_CRON .....	379	MC2 .....	<b>8</b>
LOG_DAEMON .....	379	MC3 .....	<b>8</b>
LOG_DEBUG .....	380	MCL_CURRENT .....	343
LOG_EMERG .....	380	MCL_FUTURE .....	343
LOG_ERR .....	380	mcontext_t .....	<b>401</b>
LOG_INFO .....	380	memory mapped files .....	65
LOG_KERN .....	379	memory object .....	65
LOG_LOCAL .....	379	memory synchronization .....	100
LOG_LPR .....	379	memory-resident .....	65
LOG_MAIL .....	379	message .....	65
LOG_MASK .....	379	message catalog .....	66
LOG_NDELAY .....	379	message catalog descriptor .....	66
LOG_NEWS .....	379	message queue .....	66
LOG_NOTICE .....	380	META_CHAR .....	179
LOG_NOWAIT .....	379	MF .....	<b>8</b>
LOG_ODELAY .....	379	minimum values .....	255
LOG_PID .....	379	MINSIGSTKSZ .....	306
LOG_USER .....	379	ML .....	<b>9</b>

MLR .....	9	MSG_PEEK .....	357
MM_macros .....	233	MSG_TRUNC .....	357
MM_APPL .....	233	MSG_WAITALL .....	357
MM_CONSOLE .....	233	MS_ASYNC .....	343
MM_ERROR .....	233	MS_INVALIDATE .....	343
MM_FIRM .....	233	MS_SYNC .....	343
MM_HALT .....	233	multi-character collating element .....	66
MM_HARD .....	233	mutex .....	66
MM_INFO .....	233	MUXID_ALL .....	339
MM_NOCON .....	234	MX .....	9
MM_NOMSG .....	233	M_ .....	266
MM_NOSEV .....	233	M_E .....	266
MM_NOTOK .....	233	M_LN .....	266
MM_NRECOV .....	233	M_LOG10E .....	266
MM_NULLACT .....	233	M_LOG2E .....	266
MM_NULLLBL .....	233	M_PI .....	266
MM_NULLMC .....	233	M_SQRT1_2 .....	267
MM_NULLSEV .....	233	M_SQRT2 .....	267
MM_NULLTAG .....	233	name .....	67
MM_NULLTXT .....	233	named STREAM .....	67
MM_OK .....	233	NAME_MAX .....	100, 216, 253
MM_OPSYS .....	233	NaN .....	229
MM_PRINT .....	233	NAN .....	267
MM_RECOVER .....	233	NaN (Not a Number) .....	67
MM_SOFT .....	233	NaN arguments	
MM_UTIL .....	233	mathematical functions .....	106
MM_WARNING .....	233	native language .....	67
mode .....	66	NCCS .....	383
mode_t .....	371	NDEBUG .....	209
MON .....	9	negative response .....	67
monotonic clock .....	66	negative_sign .....	143
MON_ .....	246	network .....	67
mon_decimal_point .....	142	network address .....	67
mon_grouping .....	143	network byte order .....	68, 99
mon_thousands_sep .....	143	newline character .....	68
MORECTL .....	339	NEW_TIME .....	424
MOREDATA .....	339	NGROUPS_MAX .....	254
mount point .....	66	nice value .....	68
MPR .....	9	NI_DGRAM .....	280
MQ_OPEN_MAX .....	250	NI_NAMEREQD .....	280
MQ_PRIO_MAX .....	250	NI_NOFQDN .....	279
MSG .....	9	NI_NUMERICHOST .....	279
MSGVERB .....	166	NI_NUMERICSERV .....	280
MSG_ANY .....	339	NLDLY .....	384
MSG_BAND .....	339	nlink_t .....	371
MSG_CTRUNC .....	357	NLn .....	384
MSG_DONTRROUTE .....	357	NLSPATH .....	163
MSG_EOR .....	357	NL_ARGMAX .....	260
MSG_HIPRI .....	339	NL_CAT_LOCALE .....	287
MSG_NOERROR .....	346	NL_LANGMAX .....	260
MSG_OOB .....	357	NL_MSGMAX .....	260

NL_NMAX.....	260	FR.....	7
NL_SETD.....	287	FSC.....	8
NL_SETMAX.....	260	IP6.....	8
NL_TEXTMAX.....	260	MC1.....	8
NOEXPR.....	246	MC2.....	8
NOFLSH.....	386	MC3.....	8
non-blocking.....	68	MF.....	8
non-canonical mode input processing.....	190	ML.....	9
non-spacing characters.....	68	MLR.....	9
NOSTR.....	246	MON.....	9
NUL.....	68	MPR.....	9
NULL.....	317, 325, 329, 333, 393, 410	MSG.....	9
null byte.....	69	MX.....	9
null pointer.....	69	PIO.....	10
null string.....	69	PS.....	10
null wide-character code.....	69	RS.....	10
number sign.....	69	RTS.....	10
numerical limits.....	258	SD.....	10
NZERO.....	260	SEM.....	10
n_cs_precedes.....	143	SHM.....	11
n_sep_by_space.....	143	SIO.....	11
n_sign_posn.....	144	SPI.....	11
OB.....	9	SPN.....	11
object file.....	69	SS.....	11
OCRNL.....	384	TCT.....	11
octet.....	69	TEF.....	11
OF.....	10	THR.....	12
offset maximum.....	69	TMO.....	12
off_t.....	371	TMR.....	12
OFILL.....	384	TPI.....	12
OH.....	10	TPP.....	12
OLD_TIME.....	424	TPS.....	12
ONLCR.....	384	TRC.....	12
ONLRET.....	384	TRI.....	12
ONOCR.....	384	TRL.....	13
opaque address.....	69	TSA.....	13
open file.....	70	TSF.....	13
open file description.....	70	TSH.....	13
OPEN_MAX.....	250, 302	TSP.....	13
operand.....	70	TSS.....	13
operator.....	70	TYM.....	13
OPOST.....	384	UP.....	14
option.....	70	XSR.....	14
ADV.....	6	option-argument.....	70
AIO.....	6	options.....	
BAR.....	7	shell and utilities.....	29
BE.....	7	system interfaces.....	29
CD.....	7	ORD_CHAR.....	179
CPT.....	7	orientation.....	70
CS.....	7	orphaned process group.....	70
FD.....	7	output devices.....	185

O_constants		
defined in <fcntl.h> .....	223-224	
O_ACCMODE .....	224	
O_APPEND .....	223	
O_CREAT .....	223	
O_DSYNC .....	223	
O_EXCL .....	223	
O_NOCTTY .....	223	
O_NONBLOCK .....	223	
O_RDONLY .....	224	
O_RDWR .....	224	
O_RSYNC .....	224	
O_SYNC .....	224	
O_TRUNC .....	223	
O_WRONLY .....	224	
page .....	71	
page size .....	71	
PAGESIZE .....	250	
PAGE_SIZE .....	251	
parameter .....	71	
PARENB .....	386	
parent directory .....	71	
parent process .....	71	
parent process ID .....	71	
PARMRK .....	384	
PARODD .....	386	
PATH .....	166	
path prefix .....	72	
pathname .....	72	
pathname component .....	72	
pathname resolution .....	100	
pathname variable values .....	252	
PATH_MAX .....	253, 261	
pattern .....	72	
period .....	72	
permissions .....	72	
persistence .....	72	
pid_t .....	371	
PIO .....	10	
pipe .....	73	
PIPE_BUF .....	253	
PM_STR .....	246	
pole error .....	106	
POLLERR .....	288	
pollfd .....	288	
POLLHUP .....	288	
POLLIN .....	288	
polling .....	73	
POLLNVAL .....	288	
POLLOUT .....	288	
POLLPRI .....	288	
POLLRDBAND .....	288	
POLLRDNORM .....	288	
POLLWRBAND .....	288	
POLLWRNORM .....	288	
POLL_ERR .....	308	
POLL_HUP .....	308	
POLL_IN .....	308	
POLL_MSG .....	308	
POLL_OUT .....	308	
POLL_PRI .....	308	
portable character set .....	73, 113	
portable filename character set .....	73, 98	
positional parameter .....	73	
positive_sign .....	143	
POSIX		
conformance .....	17	
POSIX locale .....	124	
POSIX shell and utilities .....	20	
POSIX system interfaces		
conformance .....	18	
POSIX2_CHAR_TERM .....	20, 29	
POSIX2_C_DEV .....	20, 29	
POSIX2_FORT_DEV .....	20, 29	
POSIX2_FORT_RUN .....	20, 29	
POSIX2_LOCALEDEF .....	20, 30	
POSIX2_PBS .....	20, 30	
POSIX2_PBS_ACCOUNTING .....	20, 30	
POSIX2_PBS_CHECKPOINT .....	30	
POSIX2_PBS_LOCATE .....	20, 30	
POSIX2_PBS_MESSAGE .....	20, 30	
POSIX2_PBS_TRACK .....	20, 30	
POSIX2_SW_DEV .....	20, 30	
POSIX2_UPE .....	20, 30	
POSIX_ALLOC_SIZE_MIN .....	253	
POSIX_FADV_DONTNEED .....	224	
POSIX_FADV_NOREUSE .....	224	
POSIX_FADV_NORMAL .....	224	
POSIX_FADV_RANDOM .....	224	
POSIX_FADV_SEQUENTIAL .....	224	
POSIX_FADV_WILLNEED .....	224	
POSIX_MADV_DONTNEED .....	344	
POSIX_MADV_NORMAL .....	344	
POSIX_MADV_RANDOM .....	344	
POSIX_MADV_SEQUENTIAL .....	344	
POSIX_MADV_WILLNEED .....	344	
POSIX_REC_INCR_XFER_SIZE .....	253	
POSIX_REC_MAX_XFER_SIZE .....	253	
POSIX_REC_MIN_XFER_SIZE .....	253	
POSIX_REC_XFER_ALIGN .....	253	
POSIX_TYPED_MEM_ALLOCATE .....	344	
POSIX_TYPED_MEM_ALLOCATE_CONTIG .....	344	

POSIX_TYPED_MEM_MAP_ALLOCATABLE	344	PTHREAD_KEYS_MAX	251
preallocation	73	PTHREAD_MUTEX_DEFAULT	290
preempted process (or thread)	74	PTHREAD_MUTEX_ERRORCHECK	290
previous job	74	PTHREAD_MUTEX_INITIALIZER	290
printable character	74	PTHREAD_MUTEX_NORMAL	290
printable file	74	PTHREAD_MUTEX_RECURSIVE	290
priority	74	PTHREAD_ONCE_INIT	290
priority band	74	PTHREAD_PRIO_INHERIT	290
priority inversion	74	PTHREAD_PRIO_NONE	290
priority scheduling	74	PTHREAD_PRIO_PROTECT	290
priority-based scheduling	75	PTHREAD_PROCESS_PRIVATE	290
PRIO_ constants		PTHREAD_PROCESS_SHARED	290
defined in <sys/resource.h>	347	PTHREAD_RWLOCK_INITIALIZER	290
PRIO_PGRP	347	PTHREAD_SCOPE_PROCESS	290
PRIO_PROCESS	347	PTHREAD_SCOPE_SYSTEM	290
PRIO_USER	347	PTHREAD_STACK_MIN	251
privilege	75	PTHREAD_THREADS_MAX	251
process	75	PTRDIFF_MAX	322
process group	75	PTRDIFF_MIN	322
process group ID	75	PWD	166
process group leader	75	P_ALL	377
process group lifetime	75	p_cs_precedes	143
process ID	76	P_GID	377
process ID reuse	101	P_PID	377
process lifetime	76	p_sep_by_space	143
process memory locking	76	p_sign_posn	143
process termination	76	P_tmpdir	325
process virtual time	76	quiet NaN	229
process-to-process communication	76	QUOTED_CHAR	179
program	77	radix character	77
protocol	77	RADIXCHAR	246
PROT_EXEC	343	RAND_MAX	329
PROT_NONE	343	range error	106
PROT_READ	343	result overflows	106
PROT_READ constants		result underflows	106
in <sys/mman.h>	343	read-only file system	77
PROT_WRITE	343	read-write lock	77
PS	10	real group ID	77
pseudo-terminal	77	real time	78
PTHREAD_BARRIER_SERIAL_THREAD	290	real user ID	78
PTHREAD_CANCELED	290	realtime	24
PTHREAD_CANCEL_ASYNCHRONOUS	290	REALTIME	206, 274, 298, 302
PTHREAD_CANCEL_DEFERRED	290	realtime signal extension	78
PTHREAD_CANCEL_DISABLE	290	REALTIME_THREADS	26
PTHREAD_CANCEL_ENABLE	290	realtime threads	26
PTHREAD_COND_INITIALIZER	290	record	78
PTHREAD_CREATE_DETACHED	290	redirection	78
PTHREAD_CREATE_JOINABLE	290	redirection operator	78
PTHREAD_DESTRUCTOR_ITERATIONS	251	reentrant function	78
PTHREAD_EXPLICIT_SCHED	290	referenced shared memory object	78
PTHREAD_INHERIT_SCHED	290	refresh	79



region .....	79	root directory .....	80
REGTYPE .....	381	RPROTDAT .....	338
regular expression .....	79	RPROTDIS .....	338
basic .....	171	RPROTNORM .....	338
extended .....	175	RS .....	<b>10</b>
grammar .....	179	RS_HIPRI .....	338
regular file .....	79	RTLD_GLOBAL .....	218
REG_constants		RTLD_LAZY .....	218
defined in <regex.h> .....	<b>296</b>	RTLD_LOCAL .....	218
REG_BADBR .....	296	RTLD_NOW .....	218
REG_BADPAT .....	296	RTS .....	<b>10</b>
REG_BADRPT .....	297	RTSIG_MAX .....	251, 305
REG_EBRACE .....	296	runnable process (or thread) .....	80
REG_EBRACK .....	296	running process (or thread) .....	80
REG_ECOLLATE .....	296	runtime values	
REG_ECTYPE .....	296	increasable .....	253
REG_EESCAPE .....	296	invariant .....	249
REG_ENOSYS .....	297	rusage .....	<b>347</b>
REG_EPAREN .....	296	RUSAGE_CHILDREN .....	347
REG_ERANGE .....	297	RUSAGE_SELF .....	347
REG_ESPACE .....	297	R_ANCHOR .....	179
REG_ESUBREG .....	296	R_OK .....	410
REG_EXTENDED .....	296	saved resource limits .....	80
REG_ICASE .....	296	saved set-group-ID .....	80
REG_NEWLINE .....	296	saved set-user-ID .....	80
REG_NOMATCH .....	296	SA_constants	
REG_NOSUB .....	296	declared in <signal.h> .....	306
REG_NOTBOL .....	296	SA_NOCLDSTOP .....	306
REG_NOTEOL .....	296	SA_NOCLDWAIT .....	306
relative pathname .....	79, 100	SA_NODEFER .....	306
relocatable file .....	79	SA_ONSTACK .....	306
relocation .....	79	SA_RESETHAND .....	306
requested batch service .....	79	SA_RESTART .....	306
requirements .....	17	SA_SIGINFO .....	306
result overflows .....	106	SCHAR_MAX .....	259
result underflows .....	106	SCHAR_MIN .....	259-260
RE_DUP_MAX .....	251, 254	scheduling .....	80
rlimit .....	<b>347</b>	scheduling allocation domain .....	80
RLIMIT_AS .....	348	scheduling contention scope .....	81
RLIMIT_CORE .....	347	scheduling policy .....	81, 101
RLIMIT_CPU .....	347	SCHED_FIFO .....	298
RLIMIT_DATA .....	347	SCHED_OTHER .....	298
RLIMIT_FSIZE .....	347	SCHED_RR .....	298
RLIMIT_NOFILE .....	348	SCHED_SPORADIC .....	298
RLIMIT_STACK .....	348	SCM_RIGHTS .....	356
RLIM_INFINITY .....	347	screen .....	81
RLIM_SAVED_CUR .....	347	scroll .....	81
RLIM_SAVED_MAX .....	347	SD .....	<b>10</b>
RMSGD .....	338	seconds since the Epoch .....	102
RMSGN .....	338	SEEK_CUR .....	223, 325, 412
RNORM .....	338	SEEK_END .....	223, 325, 412

SEEK_SET .....	223, 325, 412	SIGQUEUE_MAX .....	251
SEGV_ACCERR .....	308	SIGQUIT .....	305
SEGV_MAPERR .....	308	SIGRTMAX .....	304
SEM .....	10	SIGRTMIN .....	304
semaphore .....	81, 102	SIGSEGV .....	305, 308
semaphore lock operation .....	102	SIGSTKSZ .....	306
semaphore unlock operation .....	103	SIGSTOP .....	305
SEM_NSEMS_MAX .....	251	SIGSYS .....	305
SEM_UNDO .....	351	SIGTERM .....	305
SEM_VALUE_MAX .....	251	SIGTRAP .....	305, 308
session .....	81	SIGTSTP .....	305
session leader .....	82	SIGTTIN .....	305
session lifetime .....	82	SIGTTOU .....	305
SETALL .....	351	SIGURG .....	305
SETVAL .....	351	SIGUSR1 .....	305
shall .....	5	SIGUSR2 .....	305
shared memory object .....	82	SIGVTALRM .....	305
shell .....	82	SIGXCPU .....	305
SHELL .....	166	SIGXFSZ .....	305
shell script .....	82	SIG_ATOMIC_MAX .....	322
shell, the .....	82	SIG_ATOMIC_MIN .....	322
SHM .....	11	SIG_BLOCK .....	306
SHMLBA .....	353	SIG_DFL .....	304
SHM_RDONLY .....	353	SIG_ERR .....	304
SHM_RND .....	353	SIG_HOLD .....	304
should .....	5	SIG_IGN .....	304
SHRT_MAX .....	259	SIG_SETMASK .....	306
SHRT_MIN .....	260	SIG_UNBLOCK .....	306
SHUT_RD .....	357	single-quote .....	83
SHUT_RDWR .....	358	SIO .....	11
SHUT_WR .....	358	SIZE_MAX .....	322
SIGABRT .....	305	size_t .....	333, 371
SIGALRM .....	305	SI_ASYNCIO .....	308
SIGBUS .....	305, 308	SI_MSGQ .....	308
SIGCHLD .....	305, 308	SI_QUEUE .....	308
SIGCONT .....	305	SI_TIMER .....	308
SIGEV_NONE .....	304	SI_USER .....	308
SIGEV_SIGNAL .....	304	slash .....	83
SIGEV_THREAD .....	304	SNDZERO .....	339
SIGFPE .....	305, 308	socket .....	83
SIGHUP .....	305	socket address .....	83
SIGILL .....	305, 308	SOCK_DGRAM .....	356
siginfo_t .....	307	SOCK_RAW .....	356
SIGINT .....	305	SOCK_SEQPACKET .....	356
SIGKILL .....	305	SOCK_STREAM .....	356
signal .....	82	soft limit .....	83
signal stack .....	83	SOL_SOCKET .....	356
signaling NaN .....	229	SOMAXCONN .....	357
SIGPIPE .....	305	source code .....	83
SIGPOLL .....	305, 308	SO_ACCEPTCONN .....	356
SIGPROF .....	305	SO_BROADCAST .....	357

SO_DEBUG	357	string	85
SO_DONTROUTE	357	striotcl	336
SO_ERROR	357	strpeek	336
SO_KEEPALIVE	357	strrecvfd	336
SO_LINGER	357	str_list	336
SO_OOINLINE	357	str_mlist	337
SO_RCVBUF	357	ST_NOSUID	365
SO_RCVLOWAT	357	ST_RDONLY	365
SO_RCVTIMEO	357	subprofiling	22
SO_REUSEADDR	357	subshell	86
SO_SNDBUF	357	successfully transferred	86
SO_SNDLOWAT	357	supplementary group ID	86
SO_SNDTIMEO	357	suseconds_t	371
SO_TYPE	357	suspended job	86
space character	84	symbolic link	86
spawn	84	SYMLINK_MAX	253, 261
special built-in	84	SYMLOOP_MAX	251
special parameter	84	SYMTYPE	381
SPEC_CHAR	180	synchronized I/O completion	86
SPI	11	synchronized I/O data integrity completion	87
spin lock	84	synchronized I/O file integrity completion	87
SPN	11	synchronized I/O operation	87
sporadic server	84	synchronized input and output	86
SS	11	synchronous I/O operation	87
SSIZE_MAX	259, 372	synchronously-generated signal	87
ssize_t	371	system	87
SS_DISABLE	306	system console	88
SS_ONSTACK	306	system crash	88
SS_REPL_MAX	251	system databases	88
stack_t	307	system documentation	88
standard error	84	system process	88
standard input	84	system reboot	88
standard output	84	system trace event	88
standard utilities	85	system-wide	89
stat data structure	360	S_ constants	
stderr	325	defined in <sys/stat.h>	360-361
STDERR_FILENO	415	S_ macros	
stdin	325	defined in <sys/stat.h>	361
STDIN_FILENO	415	S_BANDURG	338
stdout	326	S_ERROR	338
STDOUT_FILENO	415	S_HANGUP	338
strbuf	336	S_HIPRI	338
STREAM	85	S_IFBLK	361
stream	85	S_IFCHR	361
STREAM	220	S_IFDIR	361
STREAM end	85	S_IFIFO	361
STREAM head	85	S_IFLNK	361
STREAMS	28, 336	S_IFMT	360
STREAMS multiplexor	85	S_IFREG	361
STREAM_MAX	251	S_IFSOCK	361
strfdinsert	336	S_INPUT	338

S_IRGRP .....	361	terminal types.....	185
S_IROTH .....	361	termios .....	187
S_IRUSR .....	361	canonical mode input processing .....	189
S_IRWXG .....	361	control modes.....	196
S_IRWXO .....	361	controlling terminal .....	188
S_IRWXU .....	361	input modes.....	193
S_ISBLK .....	361	local modes .....	197
S_ISCHR .....	362	non-canonical mode input processing.....	190
S_ISDIR .....	362	output modes .....	194
S_ISFIFO .....	362	special control characters .....	198
S_ISGID .....	361-362	text column .....	89
S_ISLNK .....	362	text file.....	89
S_ISREG .....	362	TGEXEC.....	381
S_ISSOCK .....	362	TGREAD.....	381
S_ISUID .....	361-362	TGWRITE.....	381
S_ISVTX .....	361	THOUSEP .....	246
S_IWGRP .....	361	THR .....	12
S_IWOTH .....	361	thread .....	89
S_IWUSR .....	361	thread ID.....	90
S_IXGRP .....	361	thread list.....	90
S_IXOTH .....	361	thread-safe.....	90
S_IXUSR .....	361	thread-safety.....	103
S_MSG .....	338	thread-specific data key .....	90
S_OUTPUT .....	338	tilde .....	90
S_RDBAND .....	338	timeb.....	369
S_RDNORM .....	338	timeouts .....	90
S_TYPEISMQ .....	362	timer .....	90
S_TYPEISSEM .....	362	timer overrun.....	90
S_TYPEISSHM .....	362	TIMER_ABSTIME.....	393
S_TYPEISTMO .....	362	TIMER_MAX.....	252
S_WRBAND .....	338	timer_t.....	371
S_WRNORM .....	338	timeval.....	349, 367
tab character .....	89	time_t .....	371
TABDLY .....	384	TMAGIC.....	381
TABn.....	384	TMAGLEN.....	381
TCIFLUSH .....	387	TMO .....	12
TCIOFF .....	387	TMPDIR.....	166
TCIOFLUSH .....	387	TMP_MAX .....	325
TCION .....	387	TMR .....	12
TCOOFF .....	387	TOEXEC .....	381
TCOON .....	387	token.....	91
TCP_NODELAY .....	286	TOREAD.....	381
TCSADRAIN .....	386	TOSTOP.....	386
TCSAFLUSH .....	386	TOWRITE.....	381
TCSANOW .....	386	TPI.....	12
TCT .....	11	TPP.....	12
TEF .....	11	TPS.....	12
TERM .....	166	trace analyzer process.....	91
terminal		trace controller process .....	91
controlling.....	188	trace event.....	91
terminal (or terminal device) .....	89	trace event type.....	91

trace event type mapping .....	91	UINT_FASTN_MAX .....	321
trace filter .....	91	UINT_LEASTN_MAX .....	321
trace generation version .....	91	UINT_MAX .....	259
trace log .....	91	ULLONG_MAX .....	260
trace point .....	92	ULONG_MAX .....	259
trace stream .....	92	UL_GETFSIZE .....	402
trace stream identifier .....	92	UL_SETFSIZE .....	402
trace system .....	92	unbind .....	93
traced process .....	92	undefined .....	6
TRACE_EVENT_NAME_MAX .....	252	unspecified .....	6
TRACE_NAME_MAX .....	252	UP .....	14
TRACE_SYS_MAX .....	252	upper multiplexing .....	85
TRACE_USER_EVENT_MAX .....	252	upshifting .....	93
tracing .....	27, 103	user database .....	93
tracing status of a trace stream .....	92	user ID .....	93
TRAP_BRKPT .....	308	user name .....	93
TRAP_TRACE .....	308	user trace event .....	94
TRC .....	12	USER_PROCESS .....	424
TRI .....	12	USHRT_MAX .....	259
TRL .....	13	utility .....	94, 107
TSA .....	13	Utility Syntax Guidelines .....	203
TSF .....	13	utmpx .....	424
TSGID .....	381	variable .....	94
TSH .....	13	variable assignment .....	107
TSP .....	13	VEOF .....	383
TSS .....	13	VEOL .....	383
TSUID .....	381	VERASE .....	383
TSVTX .....	381	vertical-tab character .....	94
TTY_NAME_MAX .....	252	VFS .....	365
TUEXEC .....	381	VINTR .....	383
TUREAD .....	381	VKILL .....	383
TUWRITE .....	381	VQUIT .....	383
TVERSION .....	381	VSTART .....	383
TVERSLEN .....	381	VSTOP .....	383
TYM .....	13	VSUSP .....	383
typed memory name space .....	92	VTDLY .....	385
typed memory object .....	92	VTn .....	385
typed memory pool .....	92	warning	
typed memory port .....	93	OB .....	9
TZ .....	166	OF .....	10
TZNAME_MAX .....	252	WCHAR_MAX .....	322, 428
T_FMT .....	246	WCHAR_MIN .....	322, 428
T_FMT_AMPM .....	246	WCONTINUED .....	377
t_scalar_t .....	336	WEOF .....	426, 428, 430
t_uscalar_t .....	336	WEXITED .....	377
UCHAR_MAX .....	259	WEXITSTATUS .....	329, 377
ucontext_t .....	401	white space .....	94
uid_t .....	371	wide characters .....	117
UINTMAX_MAX .....	322	wide-character code (C language) .....	94
UINTN_MAX .....	321	wide-character input/output functions .....	94
UINTPTR_MAX .....	322	wide-character string .....	95

WIFCONTINUED .....	377
WIFEXITED .....	329, 377
WIFSIGNALED .....	329, 377
WIFSTOPPED .....	329, 377
WINT_MAX .....	322
WINT_MIN .....	322
WNOHANG .....	329, 377
WNOWAIT .....	377
word .....	95
WORD_BIT .....	258-259
working directory .....	95
worldwide portability interface .....	95
WRDE_APPEND .....	432
WRDE_BADCHAR .....	432
WRDE_BADVAL .....	432
WRDE_CMDSUB .....	432
WRDE_DOOFFS .....	432
WRDE_NOCMD .....	432
WRDE_NOSPACE .....	432
WRDE_NOSYS .....	432
WRDE_REUSE .....	432
WRDE_SHOWERR .....	432
WRDE_SYNTAX .....	432
WRDE_UNDEF .....	432
write .....	95
WSTOPPED .....	377
WSTOPSIG .....	329, 377
WTERMSIG .....	329, 377
WUNTRACED .....	329, 377
W_OK .....	410
XSI .....	14, 95
conformance .....	17
XSI conformance .....	21
XSI options groups .....	24
XSI STREAMS .....	28
XSI system interfaces	
conformance .....	21
XSI-conformant .....	95
XSR .....	14
X_OK .....	410
YESEXPR .....	246
YESSTR .....	246
zombie process .....	96